Universidad Nacional Autónoma de México

Facultad de Ingeniería

T E S I S

Diseño de un FRAMEWORK para aplicaciones WEB

PARA OBTENER EL TÍTULO DE INGENIERO EN COMPUTACIÓN

PRESENTAN:

- ALARCÓN BÁRCENAS NORMA
- BECERRIL CRUZ ARACELI
- OLANDES HERNÁNDEZ MARISOL
- VELASCO OLVERA TOMASA MARGARITA
- VILLANUEVA ROSALES MARCO ANTONIO

DIRECTOR: ING. LUCILA PATRICIA ARELLANO MENDOZA

CIUDAD UNIVERSITARIA, MÉXICO D.F. NOVIEMBRE 2006

Dedicatorias

Quiero agradecer a mis padres todo el apoyo, constancia y dedicación que me brindaron de manera incondicional para mi desarrollo personal y profesional, por lo que estaré eternamente agradecida y que gracias a ellos termino una fase más en mi vida.

También, quiero agradecer a las personas que me han brindado su apoyo incondicional, que han sufrido y gozado conmigo cada uno de mis fracasos y logros, por quienes siento un gran amor, respeto y admiración, mis hermanos.

Norma Alarcón Bárcenas

Agradezco a ti Papá por el apoyo incondicional que siempre me brindaste, por tu confianza y fé que me mantuvo firme en esta etapa que concluye y de la cuál tú fuiste una base importante.

A ti Mamá por tu amor, tu espíritu de lucha, tu apoyo y tu gran ejemplo de vida que inspira a seguir adelante aún en las situaciones más difíciles.

A mis hermanos y sobrinos por ser mi gran motor, por ser la motivación para superarme, por llenarme de alegría y fuerza suficiente para afrontar lo bueno y malo que pude haber pasado para llegar a este momento.

Araceli Becerril Cruz

A Dios.
Por haberme permitido llegar hasta este punto, por los triunfos y los momentos difíciles que me han enseñado a valorarte cada día más.

A mi madre.
Por haberme apoyado en todo momento, por sus consejos, sus valores, su confianza, por la motivación constante que me ha permitido ser una persona de bien, pero más que nada, por su amor y comprensión . Eres una gran Amiga te quiero mucho.

A mis familiares.
A mis hermanos Isael y Said, a mi abuelita, a mi cuñada Ericka y a mis sobrinos Marisol y Said gracias por su apoyo y motivación en todo momento.

A Héctor Juárez.
Por tu apoyo, tu confianza y tu motivación para la realización de esta tesis. Pero principalmente gracias por tu amistad "TE EXTRAÑO".

A mis maestros.
A todos los maestros que a lo largo de mis estudios me brindaron su apoyo y amistad. Gracias por su tiempo, por su apoyo así como por la sabiduría que me transmitieron en el desarrollo de mi formación profesional.

Marisol Olandes Hernández

Hoy culmino una de las etapas más importantes de mi vida y quiero compartirla con aquellos seres especiales que me han permitido estar a su lado y diariamente me reiteran su cariño, su amistad y sobre todo su apoyo para salir siempre adelante.....
No me queda mas que darles las gracias.....

Gracias a ti mamá
Por enseñarme a luchar por mis sueños, por tus sacrificios para forjarme como una mujer de bien, por tu fortaleza ante problemas difíciles, por tus desvelos para cuidarme, por quererme tal y como soy pero sobre todo gracias por existir porque eres la mejor lección que me puede dar la vida.

Gracias a todos mis hermanos
Por compartir momentos de felicidad y de tristeza conmigo, por quererme y hacerme reflexionar siempre. Cada uno de ustedes me ha enseñado muchas cosas, a su lado he aprendido el deseo de superación y no cerrarse las puertas nunca. Han sido una muestra de ejemplo a seguir.

Gracias a mis amigos
Por apoyarme incondicionalmente, por ayudarme en momentos difíciles, por estar a mi lado.

Gracias UNAM
Por siempre tener un lugar para mi, por dejarme aprender en tus aulas, por enseñarme a valorar lo bueno. Gracias a tu apoyo porque sin ti no hubiera podido alcanzar mi gran sueño. Gracias a todos mis maestros por compartir su sabiduría.

Gracias a ti papá
Por tus sabios consejos, tu tiempo para escucharme, tu manera de regañarme para hacer las cosas bien, gracias por tu confianza y por creer en mi.....GRACIAS POR CONSENTIRME.

Gracias a ti Lui
Por tu amor, tu apoyo incondicional, tu confianza, por impulsarme a ser cada día mejor. Eres mi sueño hecho realidad.....TE AMO MI AMOR!

Gracias a ti Caro
Por nacer, por ser uno de los motivos de mi existencia, por tu ternura, tu inteligencia. Me has enseñado mucho a pesar de tu corta edad y quiero decirte que siempre estaré orgullosa de tenerte a mi lado. Eres una niña encantadora...eres la felicidad de nuestro hogar.....TQM.

Gracias a nuestra directora de tesis, por su tiempo y su paciencia para dirigirnos en el desarrollo del presente trabajo.

Y a todas esas personas que aun sin nombrarlas saben que ocupan un lugar especial en mi corazón porque me han acompañado a lo largo de mi camino y han confiado en mi y me han brindado un hombro para llorar, una mano para continuar, palabras de aliento para no desertar.

Gracias.

Margarita Velasco Olvera

A mi mamá y a mi papá les agradezco el constante apoyo incondicional y cariño que me han brindado a lo largo de toda mi vida, sin ustedes no hubiera logrado llegar a este punto ¡Los amo!

A mis hermanos Víctor y Elena, mis sobrinos Toño, Beto y Carlitos y a mi cuñada Dalia, por estar siempre dándome ánimos de seguir y siempre estar al cuidado o pendientes de mí ¡Los quiero mucho!

A mi madrina Gloria, mis primos Fabiola, Alma y Manuel por estar siempre pendientes de mí y por su cariño ¡Siempre estaremos juntos!

A mis amigas de tesis Ara, Mago, Marisol y Norma por colaborar en este trabajo, ¡Hasta que por fin nos decidimos y lo logramos!

A Ana García y Colomé a quien le agradezco sus consejos y orientación en el momento en que más lo necesitaba

A Héctor Juárez (en memoria) y Miguel Rivas por su apoyo y motivación profesional que nos ayudó en la realización de este trabajo.

A mis amigos Claus, Ara, Sofi, Erik, Gaby, Víctor, Chio y Paty por estar siempre conmigo en las buenas y las malas y por saber escucharme.

A mis maestros de la UNAM por transmitirnos con empeño y dedicación los conocimientos necesarios para nuestro desarrollo profesional y personal y por crearnos la conciencia y sensibilidad de que nuestro país requiere de nuestra colaboración y ética para que sobresalga.

A todas aquellas personas que han iluminado mi camino y que de algún modo han contribuido en mi enriquecimiento personal y profesional.

Por mi raza hablará el espíritu

Marco Antonio Villanueva Rosales

Índice

PRÓLOGO

En el desarrollo del presente proyecto se exponen los elementos y las herramientas necesarias que se requieren para el diseño de un Framework que pueda adaptarse de manera rápida y sencilla en cualquier área de desarrollo, así como también se muestra la importancia que se tiene al poder construir un Framework propio.

La idea de diseñar un Framework surgió a partir de las necesidades que se tienen en la actualidad, de mejorar la calidad de los sistemas, disminuir el tiempo de desarrollo, adaptar componentes existentes y el costo de las soluciones basadas en computadoras.

A medida que se agrava el problema del tiempo para el desarrollo de software, los desarrolladores tienen que ocuparse cada día más de la tarea de evaluar el tiempo que se llevan en desarrollar un nuevo sistema.
El tema es prácticamente nuevo y son pocos los libros de texto o manuales disponibles sobre el mismo.
Para la realización de este trabajo se llevó a cabo una investigación de campo (recolección de información y conocimientos previos) y una documental.
Dentro del Capítulo I se plantean los objetivos y se expone la problemática que se tiene en la actualidad con los sistemas de computadoras y por que es importante el desarrollo de un Framework.

En el Capítulo II se explican algunos modelos para el desarrollo de Software así como ventajas y desventajas de cada uno de ellos. Características del modelo orientado a objetos tales como la abstracción, encapsulación, modularidad y jerarquía. También se plantean los fundamentos de UML, que es el Lenguaje de Modelado Unificado que sirve para modelar, diseñar, estructurar y documentar el software antes de lanzarse a programar; es el medio de visualizar un diseño y

comprobar que cumple todos los requisitos para él estipulados. En este capitulo también se hace un análisis de las herramientas Web para el desarrollo de software tanto comerciales como open source. Se mencionan algunos patrones de diseño los cuales son muy importantes para la reutilización de código. Descripción de framework y las etapas que hay que seguir para desarrollarlo.

En el Capítulo III se muestra una tabla comparativa de las alternativas de solución, adquisición de un framework comercial, usar un framework de código abierto o construir un framework.

El posible resultado de la propuesta, se analiza en la última parte (Capítulo IV) de la tesis junto con las conclusiones acerca del diseño propuesto.

1 ANTECEDENTES

1.1 Introducción

Mejorar la calidad y reducir el costo de las soluciones basadas en computadoras es el principal desafío en el desarrollo de software. Una manera de ayudar a cumplir con este objetivo es maximizar el rehúso y posibilidad de evolución. El rehúso produce reducción de tiempo y costo e incrementos de calidad, en la medida que el desarrollador pueda encontrar, utilizar y adaptar al nuevo contexto, aquellas soluciones que ya han sido probadas y usadas exitosamente. Se puede hacer rehúso de un dominio del problema, de una clase, de un componente, de un diseño, de mecanismos, de una idea o de un patrón.

Un patrón es una idea que ha sido útil en un contexto y probablemente lo sea en otros. Es un modo de proveer información en forma de una declaración de problema, algunas restricciones, una presentación de una solución ampliamente aceptada al problema, y luego una discusión de las consecuencias de esa solución.

Los patrones ayudan a aliviar la complejidad del software en varias fases en el ciclo de vida. Capturan ideas y soluciones obtenidas por experiencia y las hacen accesible para los desarrolladores, analistas e ingenieros menos expertos o principiantes.

Las etapas de análisis y diseño son fundamentales en el desarrollo de software. En la fase de análisis se estudia y especifica el dominio del problema dentro del contexto de objetivos y metas preestablecidas, buscando comprender los detalles y complejidades concernientes al problema para ir hacia un modelo mental de lo que se quiere alcanzar. A menudo encontramos que muchos aspectos de un problema en un dominio ya han aparecido en otros proyectos diferentes. Abstraer el modelo conceptual y aplicarlos al problema en cuestión es el desafío. De esto tratan los patrones.

Para contar con herramientas flexibles, manejables, extendibles, fáciles de modificar y mantener, es necesario diseñar aplicando sistemáticamente los principios de la ingeniería de software moderna. La tecnología orientada a objetos ha demostrado ser una herramienta muy poderosa para resolver problemas de gran envergadura y complejidad, que requieren alto grado de integridad en la información y facilidades para la extensión y evolución.

Es la manera más efectiva de lograr reusabilidad, pues permite definir mediante abstracción y composición, los diferentes componentes de una aplicación, que son conectados para lograr el comportamiento esperado del sistema.

Siguiendo con esta idea, en el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, librerías y un lenguaje de scripting entre otros tipos de software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Un framework representa una arquitectura de software que modela las relaciones generales de las entidades del dominio, para lograr los beneficios de reusabilidad, extensibilidad y mantenibilidad.

1.2 Planteamiento del problema

En nuestros días la dinámica de las organizaciones dentro de un entorno cambiante exige al área de tecnología de información la capacidad de obtener mejores resultados en menor tiempo. Una parte importante de la tecnología de información son los sistemas o aplicaciones que sustentan los procesos de negocio. Existen diversas opiniones de los caminos o estrategias a seguir para lograr el desarrollo de aplicaciones que sean fácilmente adaptables a las demandas del entorno.

Es cierto que no es conveniente comenzar de base cero el desarrollo o implantación de una solución, sin embargo, existe una balanza de tiempo y costo que se debe tomar en cuenta, en la figura 1.1 hacemos una representación gráfica de este concepto, en donde se aprecia que en la medida mientras el paquete que se adquiera tenga más completitud o abarque más funcionalidad el tiempo en que se espera obtener la solución lista se reduce pero con un costo cada vez mayor, y por el otro lado si no se cuenta con un paquete el costo y tiempo de desarrollo se dispara y aunque no podemos eliminar el costo del desarrollo aún teniendo un paquete con alto grado de completitud, sí observamos que el costo de desarrollo se reduce de forma asintótica.

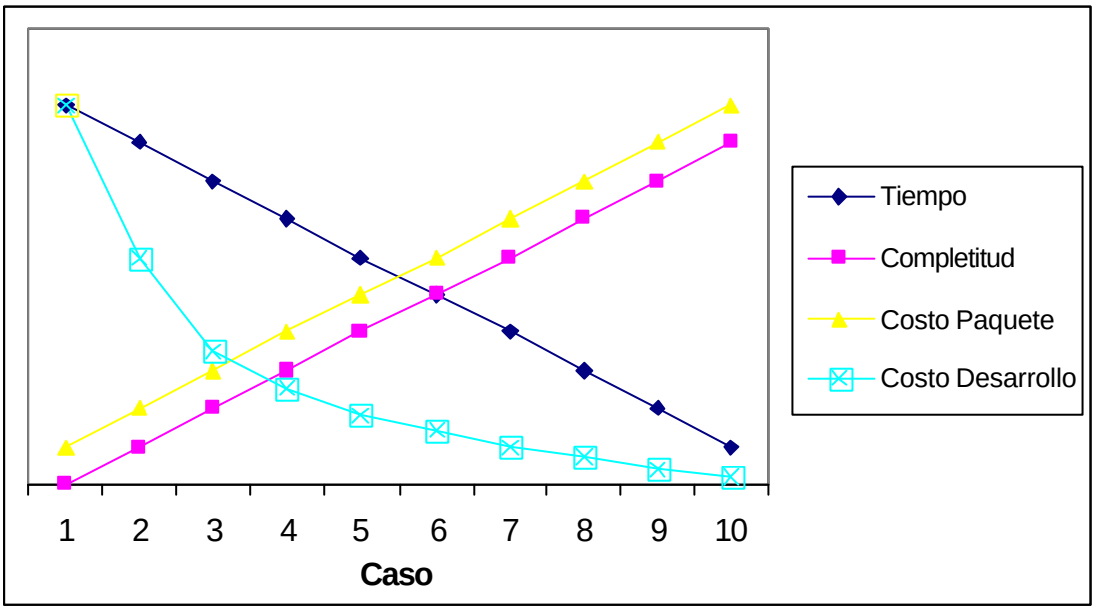


Figura 1.1 Estimaciones de costo y tiempo conforme se adquiera un paquete con más funcionalidad, el costo se dispara pero el tiempo y costo de desarrollo se reduce.

Un paquete se puede considerar desde un conjunto de librerías que los desarrolladores pueden utilizar en la construcción de un sistema de información, o un paquete que cumple con una función específica, o hasta un sistema integral de información que soporte la operación completa de una gran empresa, estos últimos paquetes son conocidos como Planeación de Recursos Empresariales

(ERP por sus siglas en inglés) y su costo está en el orden de los millones de dólares.

Aunque parezca lo contrario, elegir el esquema que se utilizará al momento de brindar una solución tecnológica no es sencillo, es decir, que porcentaje debo de cubrir con un paquete y qué porcentaje con desarrollo propio obedece únicamente a los beneficios que se obtendrán del sistema cuando esté operando. Lo que generalmente implica cierto grado de dificultad es cuantificar estos beneficios.

Dentro de los diversos esquemas que se pueden adoptar para implantar una solución tecnológica es evidente que el proceso de desarrollo o codificación de programas está siempre presente, y en cualquier caso este proceso presenta la misma problemática en mayor o menor grado. Y este trabajo se concentra principalmente en esta faceta de la implantación de soluciones.

Ahora bien este trabajo se concentra en un subconjunto de las posibles opciones y arquitecturas de desarrollo que pudiesen existir, y se puede describir como aquellos desarrollos que se realicen utilizando tecnología orientada a objetos sobre un Servidor de Aplicaciones y que cumplan con la arquitectura de establecer la frontera entre los componentes de negocio y los componentes de presentación, y además utilicen como fuente de información un manejador de base de datos relacional, soportando arquitecturas distribuidas tanto lógicas como físicas. Cabe aclarar que los sistemas informáticos con estas características tienden en la actualidad a ser la mayoría por lo que se considera que este trabajo tiene un amplio espectro de aplicación en la vida práctica.

La problemática a resolver se plantea en dos partes, la primera tiene que ver con el diseño de un FRAMEWORK que permita a los equipos de desarrollo la rápida construcción de sistemas de información minimizando la curva de aprendizaje y asegurando el cumplimiento de patrones de desarrollo.

La segunda parte tiene que ver con la inclusión de componentes dentro de este FRAMEWORK que permitan abstraer los aspectos repetitivos y también los que plantean una posibilidad de error al momento del desarrollo, para así enfocar el esfuerzo de los desarrolladores a la funcionalidad requerida para el sistema, en vez de los aspectos técnicos de su implementación.

1.3 Objetivos.

El propósito de este proyecto es diseñar un Framework para el desarrollo de aplicaciones WEB, buscando así simplificar la complejidad inherente de los desarrollos de Software en esta tecnología.

Los objetivos específicos de este proyecto son:

Mostrar las ventajas de construir un framework vs. los framework existentes en el mercado.
Diseñar un framework que:

- Cubra de manera específica y eficiente requerimientos particulares de cada sistema informático.
- Se integre de forma sencilla con otros componentes o sistemas informáticos.
- Pueda evolucionar de acuerdo a los requerimientos funcionales y/o tecnológicos que se demanden en el tiempo.
- Pueda incluir nuevas tecnologías.
- Pueda incluir y mejorar lo que sea necesario de los proyectos de código abierto a esta infraestructura.

2 MARCO TEÓRICO

2.1 Ingeniería de Software

La Ingeniería del Software es la rama de la ingeniería que aplica principios tanto de la ciencia computacional como de las matemáticas que permite desarrollar productos o soluciones correctas, utilizables y costo-efectivas a los problemas del desarrollo de software.

Al conjunto de conceptos, metodología y lenguajes que se utilizan para llevar acabo el desarrollo de algún software se le conoce como ciclo de vida del software, el cual está compuesto por cuatro fases:

Concepción (alcance del proyecto: se realiza un caso de negocio)
Elaboración (plan de proyecto: especifica las características y fundamenta la arquitectura)
Construcción (desarrolla el software)
Transición (se pone en producción el software: se transfiere al usuario final)

Existen diferentes modelos de ciclos de vida para el desarrollo de software, tales como se muestra en la figura 2.1.1.

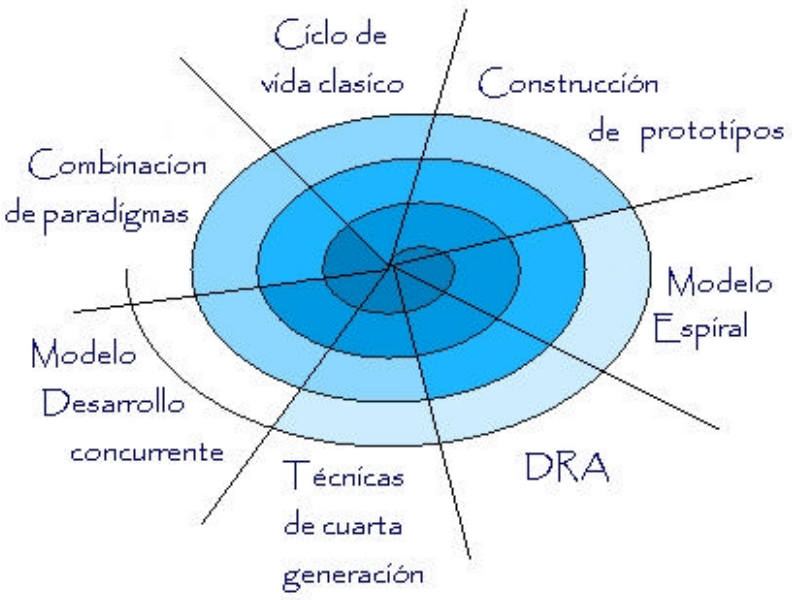


Figura 2.1.1 Fases de desarrollo de software

Sólo mencionaremos cuatro de estos modelos, los que consideramos principales.

Las ventajas y desventajas de cada uno de los modelos se presentan en la Tabla 2.1.1

Modelo	Ventajas	Desventajas
Ciclo de Vida Clásico	Este modelo es el más ampliamente usado por los ingenieros. El riesgo es bajo para los casos en que se conocen muy bien los requerimientos desde el inicio. Es fácil de aplicar.	Los proyectos reales raramente siguen éste modelo. Difícilmente se tienen claramente todos los requerimientos desde el principio. Se tienen dificultades para hacer cambios en cualquiera de las fases del ciclo. La versión funcionando del sistema no estará disponible hasta las etapas finales del desarrollo del proyecto.
Prototipo	Permite ver un software más rápido	El riesgo aumenta considerablemente cuando el

	El cliente o usuario puede evaluarlo y mejorarlo Es un modelo efectivo siempre y cuando sean bien definidas las reglas del juego desde el inicio, es decir, el técnico y el cliente deben estar de acuerdo que el prototipo es un mecanismo de definición de requerimientos.	cliente no sabe exactamente lo que quiere es decir no cuenta con la visibilidad necesaria. El cliente al ver un sistema, no distingue entre un prototipo y el sistema final, lo que genera quejas y desconciertos al usuario.
Técnicas de 4ta Generación	La reducción en el tiempo de desarrollo es dramática. Se tiene una mejora en la productividad de los desarrolladores. Se reduce el tiempo requerido para el análisis de aplicaciones pequeñas y medianas. Reduce el tiempo de implementación en aplicaciones medianas.	Su aplicación casi esta limitada a sistemas administrativos. Alto riesgo debido a la necesidad de tecnología avanzada y habilidades del grupo desarrollador. Algunas personas dicen que las herramientas de T4G no son más fáciles de utilizar que los lenguajes de programación. El código fuente producido por las herramientas es ineficiente. Existen problemas de eficiencia en algunos lenguajes. No es muy útil en aplicaciones grandes.
Espiral	Centra su atención en la reutilización de componentes y eliminación de errores en información descubierta en fases iniciales. Los objetivos de calidad son los principales. Integra desarrollo con mantenimiento. Provee un marco de desarrollo hardware /software. Es muy útil en proyectos grandes.	Difícil de administrar. Requiere experiencia en la identificación de riesgos.

Tabla 2.1.1 Ventajas y desventajas de los ciclos de vida

Es difícil decidir que modelo utilizar para el desarrollo del software, pero de manera genérica diremos que modelo utilizar según el caso.
Para sistemas bien comprendidos utiliza el Modelo de Cascada, ya que la fase de análisis de riesgos es relativamente fácil.

Si se cuenta con requerimientos estables y sistemas de seguridad críticos, utiliza Modelos Formales.

Pueden utilizarse modelos híbridos en distintas partes del desarrollo.

Posteriormente, observan que la mayoría de los programadores siguen trabajando en un lenguaje y utilizan sólo un estilo de programación. Ellos programan en un paradigma forzado por el lenguaje que utilizan. Con frecuencia no se enfrentan a métodos alternativos de resolución de un problema y por consiguiente tienen dificultad en ver la ventaja de elegir un estilo más apropiado al problema a manejar. Es por ello que aquí presentamos una alternativa más, el Modelo Orientado a Objetos.
No existe ningún estilo de programación idóneo para todas las clases de programación. La orientación a objetos se acopla a la simulación de situaciones del mundo real.

2.1.1 Modelo orientado a objetos

Los principios del modelo Orientado a Objetos son: abstracción, encapsulación, modularidad y jerarquía, fundamentalmente, y en menor grado tipificación (typing), concurrencia, persistencia.
Abstracción. Es una descripción simplificada o especificación de un sistema que enfatiza algunos de los detalles o propiedades del sistema, mientras suprime otros.

- Encapsulación. En el proceso de ocultar todos los detalles de un objeto que no contribuyen a sus características esenciales.

- Modularidad. Es la propiedad de un sistema que ha sido descompuesto en un conjunto de módulos coherentes e independientes.
- Jerarquía o herencia. Es el orden de las abstracciones organizado por niveles.
- Tipificación. Es la definición precisa de un objeto de tal forma que objetos de diferentes tipos no puedan ser intercambiados u ocasionalmente puedan intercambiarse de manera muy restringida.
- Concurrencia. Es la propiedad que distingue un objeto que está activo de uno que no lo está.
- Persistencia. Es la propiedad de un objeto a través de la cual su existencia trasciende el tiempo (es decir, el objeto continua existiendo después de que su creador ha dejado de existir) y/o el espacio (es decir, la localización del objeto se mueve del espacio de dirección en que fue creado).

Booch nos dice que para que sea un modelo orientado a objetos debe tener los cuatro primeros elementos, de lo contrario no se considera orientado a objetos.
Los beneficios con el modelo Orientado a Objetos son:
Nos ayuda a explotar el poder expresivo de todos los lenguajes de programación basados en objetos y los orientados a objetos, como Smalltalk, Object Pascal, C++, CLOS, Ada y Java .
Promueve el re-uso no solo del software, sino de diseños completos.
Produce sistemas que están construidos en formas intermedias estables y por ello son más resistentes al cambio en especificaciones y tecnología.

Pero, el principal beneficio es que da un mecanismo para formalizar el modelo de la realidad. Las relaciones entre objetos definen el comportamiento del sistema. Por ejemplo: Se dice que un objeto es Actor, si su función es operar sobre otros objetos. Si un objeto solo es manejado por otros objetos entonces es Servidor. Y es un agente si tiene ambas propiedades.

Los objetos actúan entre sí mediante mensajes, que son acciones solicitadas por el objeto transmisor para que el objeto receptor las ejecute.

El Análisis Orientado a Objetos (OOA por sus siglas en inglés de Object Oriented Analysis) "es un método de análisis que examina los requerimientos desde la perspectiva de las clases y objetos encontrados en el vocabulario del dominio del problema".

El análisis orientado a objetos se basa en un modelo de cinco capas:
Capa clase / objeto: indica las clases y objetos.
Capa de estructura: esta capa se encarga de capturar diversas estructuras de clase y objetos, tales como, las relaciones uno a muchos y la herencia.
Capa de atributos: detalla los atributos de las clases.
Capa de servicios: indica los mensajes y comportamientos del objeto (servicio y métodos).
Capa de tema: divide el diseño en unidades de implementación o asignaciones de equipos.

En el siguiente esquema de la figura 2.1.2 se pueden observar las 5 capas mencionadas.

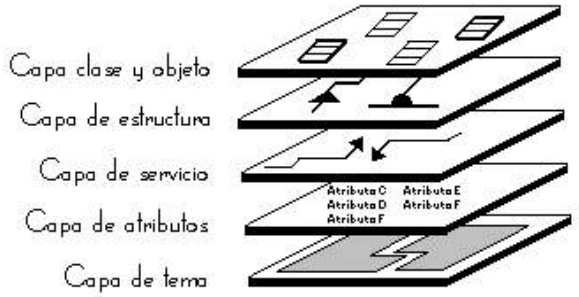


Figura 2.1.2 Capas del análisis orientado a objetos
El Diseño Orientado a Objetos "es un método de diseño abarcando el proceso de *descomposición orientado a objetos y una notación* para representar ambos

modelos lógico y físico tal como los modelos estáticos y dinámicos del sistema bajo diseño".

En cuanto a las metodologías Orientadas a Objetos, existen un gran número de métodos orientado a objetos actualmente. Muchos de los métodos pueden ser clasificados como orientados a objetos porque soportan de manera central los conceptos de la orientación a objetos. Algunas de las metodología más conocidas y estudiadas hasta antes del UML son:

- Object-Oriented Design (OOD), Booch.
- Object Modeling Technique (OMT), Rumbaugh.
- Object Oriented Analysis (OOA), Coad/Yourdon.
- Hierarchical Object Oriented Design (HOOD), ESA.
- Object Oriented Structured Design (OOSD), Wasserman.
- Object Oriented Systems Analysis (OOSA), Shaler y Mellor.
- Responsibility Driven Design (RDD), Wirfs-Brock, entre otros.

Actualmente las metodologías más importantes de análisis y diseño de sistemas han concluido en lo que se es el UML, bajo el respaldo del Object Management Group.

2.2 Fundamentos de UML

Los usuarios demandan interfaces cada vez más completas y funcionalidades más elaboradas, todo ello influyendo en el tamaño y la complejidad del producto final. Por ello, los programas deben ser estructurados de manera que puedan ser revisados, corregidos y mantenidos, rápida y eficazmente, por gente que no necesariamente ha colaborado en su diseño y construcción, permitiendo acomodar nueva funcionalidad, mayor seguridad y robustez, funcionando en todas las situaciones que puedan surgir, de manera previsible y reproducible.

Ante problemas de gran complejidad, la mejor forma de abordar la solución es modelar. Modelar es diseñar y estructurar el software antes de lanzarse a programar y es la única forma de visualizar un diseño y comprobar que cumple todos los requisitos para él estipulados, antes de que la flotilla de programadores comience a generar código. Modelando, los responsables del éxito del producto pueden estar seguros de que su funcionalidad es completa y correcta, que todas las expectativas de los usuarios finales se cumplen, que las posibles futuras ampliaciones pueden ser acomodadas, todo ello mucho antes de que la implementación haga que los cambios sean difíciles o imposibles de acomodar. Modelando, se abstraen los detalles esenciales de un problema complejo y se obtiene diseños estructurados que, además, permiten la reutilización de código, reduciendo los tiempos de producción y minimizando las posibilidades de introducir errores.

UML es un lenguaje gráfico que sirve para modelar, diseñar, estructurar, visualizar, especificar, construir y documentar software. UML proporciona un vocabulario común para toda la cadena de producción, desde quien recaba los requisitos de los usuarios, hasta el último programador responsable del mantenimiento. Es un lenguaje estándar para crear los planos de un sistema de forma completa y no ambigua.

En la siguiente figura 2.2.1 se muestra la evolución que ha tenido UML desde sus inicios.

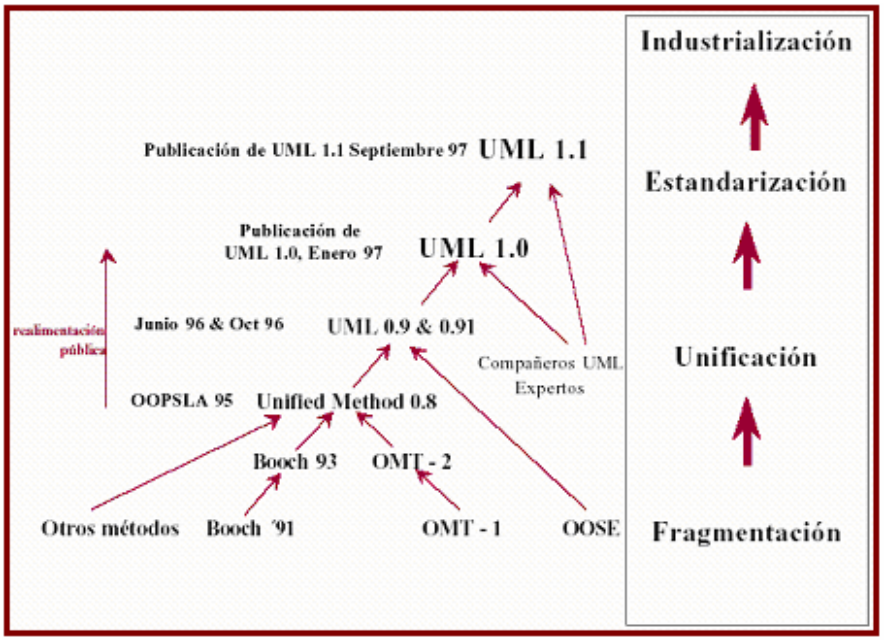


Figura 2.2.1 Evolución de UML.

2.2.1 El Lenguaje Unificado de Modelado, UML

El UML es un lenguaje de modelado cuyo vocabulario y sintaxis están ideados para la representación conceptual y física de un sistema. Sus modelos son precisos, no ambiguos, completos y pueden ser trasladados directamente a una gran variedad de lenguajes de programación, como Java, C++ o Visual Basic, pero también a tablas de bases de datos relacionales y orientadas a objetos. Es posible generar código a partir de un modelo UML (ingeniería directa) y también puede construirse un modelo a partir de la implementación (ingeniería inversa), aunque en las dos situaciones debe intervenir un mayor o menor grado de supervisión por parte del programador, en función de lo buenas que sean las herramientas empleadas.

2.2.2 Bloques básicos de construcción de UML

Los bloques básicos de construcción de UML son tres, los elementos, las relaciones y los diagramas.

Los *elementos* son abstracciones que actúan como unidades básicas de construcción. Hay cuatro tipos, los *estructurales*, los de *comportamiento*, los de *agrupación* y los de *notación*. En cuanto a los elementos estructurales son las partes estáticas de los modelos y representan aspectos conceptuales o materiales. Los elementos de comportamiento son las partes dinámicas de los modelos y representan comportamientos en el tiempo y en el espacio. Los elementos de agrupación son las partes organizativas de UML, establecen las divisiones en que se puede fraccionar un modelo. Sólo hay un elemento de agrupación, el paquete, que se emplea para organizar otros elementos en grupos. Los elementos de notación son las partes explicativas de UML, comentarios que pueden describir textualmente cualquier aspecto de un modelo. Sólo hay un elemento de notación principal, la nota.

Las *relaciones* son abstracciones que actúan como unión entre los distintos *elementos*. Hay cuatro tipos, la *dependencia*, la *asociación*, la *generalización* y la *realización*.

Los *diagramas* son la disposición de un conjunto de elementos, que representan el sistema modelado desde diferentes perspectivas. UML tiene nueve diagramas fundamentales, agrupados en dos grandes grupos, uno para modelar la estructura estática del sistema y otro para modelar el comportamiento dinámico. Los **diagramas estáticos son:** el de *clases*, de *objetos*, de *componentes* y de *despliegue*. Los **diagramas de comportamiento son:** el de *Casos de Uso*, de *secuencia*, de *colaboración*, de *estados* y de *actividades*.

2.2.3 Elementos

ELEMENTOS ESTRUCTURALES	Clase		Describe un conjunto de objetos que comparten los mismos atributos, métodos, relaciones y semántica. Las clases implementan una o más interfaces.
	Clase activa		Se trata de una clase, en la que existe procesos o hilos de ejecución concurrentes con otros elementos. Las líneas del contorno son más gruesas que en la clase “normal”
	Interfaz		Agrupación de métodos u operaciones que especifican un servicio de una clase o componente, describiendo su comportamiento, completo o parcial, externamente visible. UML permite emplear un círculo para representar las interfaces, aunque lo más normal es emplear la clase con el nombre en cursiva.
	Colaboración		Define una interacción entre elementos que cooperan para proporcionar un comportamiento mayor que la suma de los comportamientos de sus elementos.

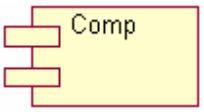
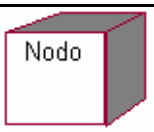
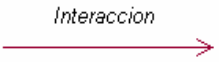
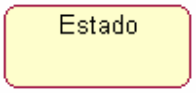
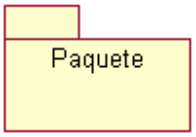
Elementos de comporta miento	Caso de uso		Describe un conjunto de secuencias de acciones que un sistema ejecuta, para producir un resultado observable de interés. Se emplea para estructurar los aspectos de comportamiento de un modelo.
	Componente		Parte física y por tanto reemplazable de un modelo, que agrupa un conjunto de interfaces, archivos de código fuente, clases, colaboraciones y proporciona la implementación de dichos elementos.
	Nodo		Elemento físico que existe en tiempo de ejecución y representa un recurso computacional con capacidad de procesar.
	Interacción		Comprende un conjunto de mensajes que se intercambian entre un conjunto de objetos, para cumplir un objetivo especifico.
	Máquinas de estados		Especifica la secuencia de estados por los que pasa un objeto o una interacción, en respuesta a eventos.
Elementos de agrupación	Paquete		Se emplea para organizar otros elementos en grupos.
Elementos de notación	Nota		Partes explicativa de UML, que puede describir textualmente cualquier aspecto del modelo

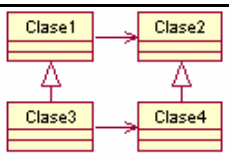
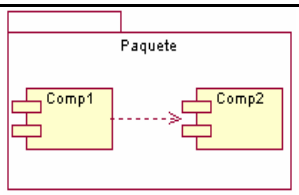
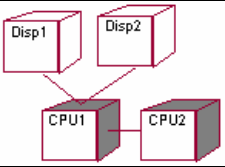
Tabla 2.2.1 Elementos de construcción en UML

2.2.4 Relaciones

Dependencia		Es una relación entre dos elementos, tal que un cambio en uno puede afectar al otro.
Asociación		Es una relación estructural que resume un conjunto de enlaces que son conexiones entre objetos.
Generalización		Es una relación en la que el elemento generalizado puede ser substituido por cualquiera de los elementos hijos, ya que comparten su estructura y comportamiento.
Realización		Es una relación que implica que la parte realizante cumple con una serie de especificaciones propuestas por la clase realizada (interfaces).

Tabla 2.2.2 Elementos de relación en UML

2.2.5 Diagramas

MODELAN ESTRUCTURA	Clases		Muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones, cubriendo la vista de diseño estática del sistema.
	Objetos		Análogo al diagrama de clases, muestra un conjunto de objetos y sus relaciones, pero a modo de vista instantánea de instancias de una clase en el tiempo.
	Componentes		Muestra la organización y dependencias de un conjunto de componentes. Cubren la vista de implementación estática de un sistema. Un componente es un módulo de código, de modo que los diagramas de componentes son los análogos físicos a los diagramas de clases.
	Despliegue		Muestra la configuración del hardware del sistema, los nodos de proceso y los componentes empleados por éstos. Cubren la vista de despliegue estática de una arquitectura.

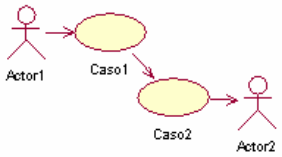
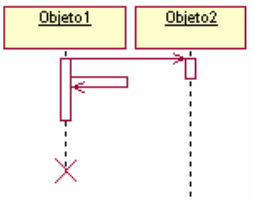
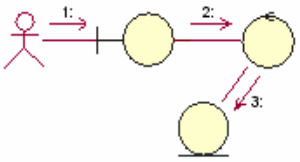
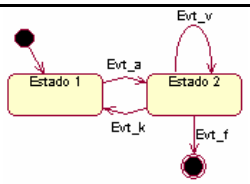
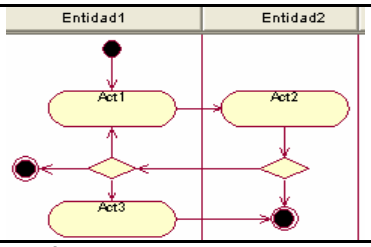
MODELAN COMPORTAMIENTO	Casos de Uso		Muestra un conjunto de casos de uso, los actores implicados y sus relaciones. Son diagramas fundamentales en el modelado y organización del sistema.
	Secuencia		Son diagramas de interacción, muestran un conjunto de objetos y sus relaciones, así como los mensajes que se intercambian entre ellos. Cubren la vista dinámica del sistema. El diagrama de secuencia resalta la ordenación temporal de los mensajes, mientras que el de colaboración resalta la organización estructural de los objetos, ambos siendo equivalentes o isomorfos. En el diagrama de colaboración de la figura de la izquierda, se puede ver que los elementos gráficos no son cajas rectangulares, como cabría esperar, y en su lugar encontramos sus versiones adornadas. Estas versiones tienen como finalidad evidenciar un rol específico del objeto siendo modelado. En la figura encontramos de izquierda a derecha y de arriba abajo un Actor, una Interfaz, un Control (modela un comportamiento) y una Instancia (modela un objeto de dato).
	Colaboración		
	Estados		Muestra una máquina de estados, con sus estados, transiciones, eventos y actividades. Cubren la vista dinámica de un sistema. Modelan comportamientos reactivos en base a eventos.
	Actividades		Tipo especial de diagrama de estados que muestra el flujo de actividades dentro de un sistema.

Tabla 2.2.3 Diagramas de UML

2.2.6 Diagrama de Clases y Diagrama de Objetos

Los diagramas de clases muestran un resumen del sistema en términos de sus clases y las relaciones entre ellas. Son diagramas estáticos que muestran **qué** es

lo que interactúa, pero no cómo interactúa o qué pasa cuando ocurre la interacción.

El siguiente diagrama modela los pedidos de un cliente a una tienda de venta por catálogo. La clase principal es “Pedido”, asociada a un cliente, una forma de pago y un conjunto de artículos. En este diagrama se muestran los diferentes tipos de relaciones así como la multiplicidad que pueden tener.

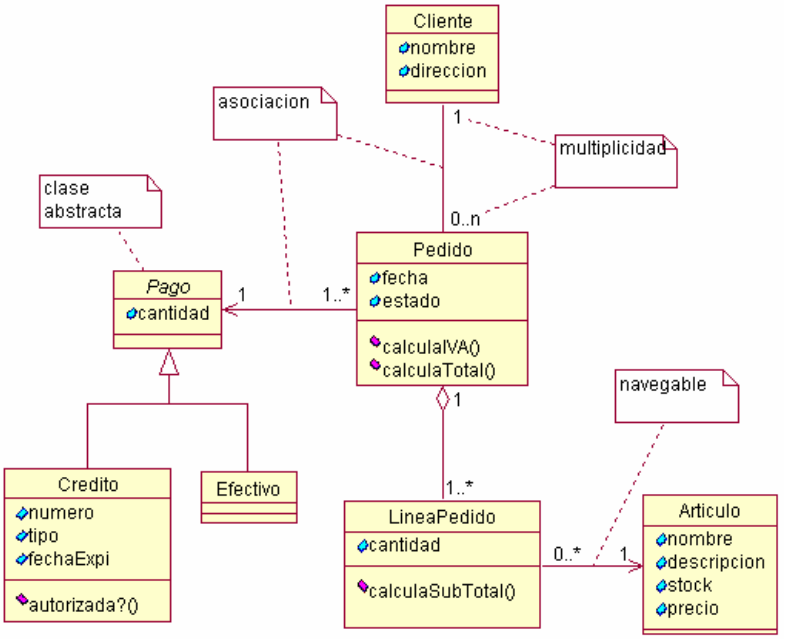


Figura 2.2.4 Diagrama de Clases

En cuanto a la multiplicidad, la siguiente tabla resume las más comunes. Hay que tener en cuenta que la multiplicidad se expresa “en el lado opuesto” de la relación y es el número de posibles instancias de una clase asociadas con una única instancia de la clase en el otro extremo.

Multiplicidad	Significado
1	Una única instancia
N / *	N instancias
0..N / 0..*	Entre ninguna y N instancias
1..N / 1..*	Entre una y N instancias
0..1	Ninguna o una instancia
N..M	Entre N y M instancias

Tabla 2.2.1 Multiplicidad en Diagramas de Clases

Los diagramas de objetos son análogos a los de clases, con la particularidad de que en lugar de encontrar clases, encontramos instancias de éstas. Son útiles para explicar partes pequeñas del modelo en las que hay relaciones complejas.

2.2.7 Diagrama de Componentes y Diagrama de Despliegue

Los componentes son módulos de código, así que los diagramas de componentes vienen a ser los análogos físicos a los diagramas de clases. Muestran como está organizado un conjunto de componentes y las dependencias que existen entre ellos.

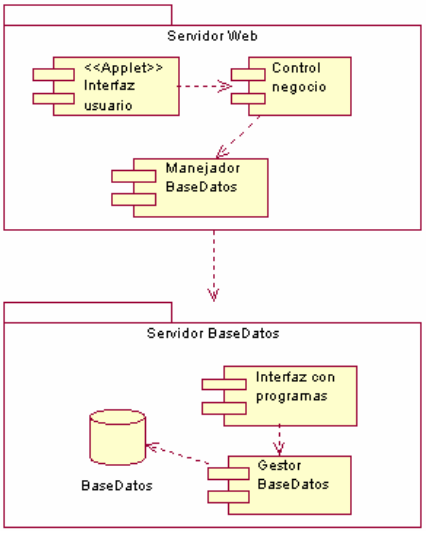


Figura 2.2.5 Diagrama de Componentes

Los diagramas de despliegue sirven para modelar la configuración hardware del sistema, mostrando qué nodos lo componen.

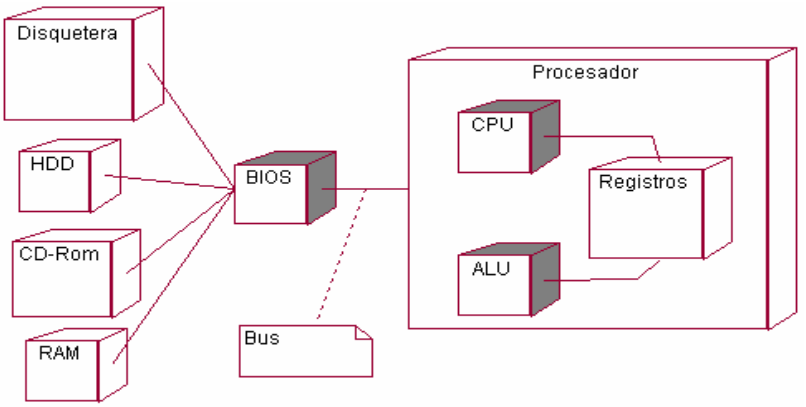


Figura 2.2.6 Diagrama de Despliegue

2.2.8 Diagrama de Casos de Uso

Los diagramas de Casos de Uso describen lo que hace un sistema desde el punto de vista de un observador externo, enfatizando el **qué** más que el cómo. Plantean escenarios, es decir, lo que pasa cuando alguien interactúa con el sistema, proporcionando un resumen para una tarea u objetivo. El siguiente Caso de Uso describe como Carlos va a desayunar (este es su objetivo), para lo que se plantea el escenario de preparar su café y el pan tostado

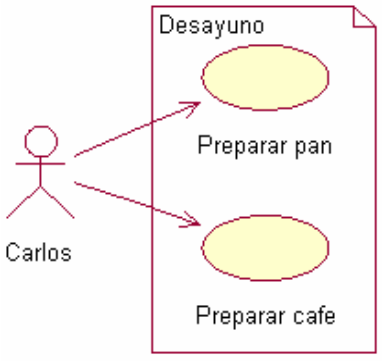


Figura 2.2.7 Diagrama de Casos de Uso nivel 1

En los Casos de Uso, los *Actores* son papeles que determinadas personas u objetos desempeñan. Se representan mediante un “hombre de palitos”, de modo

que en el ejemplo, Carlos es un Actor. Los Casos de Uso se representan por medio de *óvalos* y las líneas que unen Actores con Casos de Uso representan una asociación de comunicación.

Los Casos de Uso son acompañados por una explicación textual que clarifica las posibles carencias del lenguaje meramente gráfico. De esta manera, combinando Casos de Uso y explicación textual, se puede obtener escenarios no ambiguos, que resultan ideales en la captura de requisitos de usuario, dada su sencillez de comprensión incluso por quien no está familiarizado con UML. Los Casos de Uso se emplean también en la preparación de escenarios de pruebas con que verificar el software una vez ha sido construido.

2.2.9 Diagrama de Secuencia y Diagrama de Colaboración

Los diagramas de secuencia describen como los objetos del sistema colaboran. Se trata de un diagrama de interacción que detalla como las operaciones se llevan a cabo, qué mensajes son enviados y cuando, organizado todo en torno al tiempo. El tiempo avanza “hacia abajo” en el diagrama. Los objetos involucrados en la operación se listan de izquierda a derecha de acuerdo a su orden de participación dentro de la secuencia de mensajes.

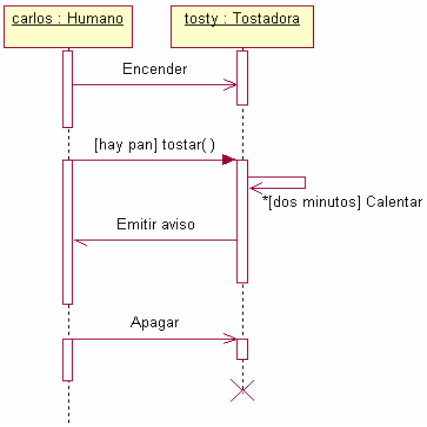


Figura 2.2.8 Diagrama de Secuencia

En el diagrama podemos observar los siguientes elementos:

Líneas verticales o “líneas de la vida” representan el tiempo de vida del objeto.

Rectángulos verticales son barras de activación y representan la duración de la ejecución del mensaje

Corchetes **“[]”** expresan condición y si están precedidos de un asterisco indican interacción mientras se cumpla la condición.

Los mensajes que son intercambiados entre los objetos de un diagrama de secuencia pueden ser *síncronos* o *asíncronos*. Los mensajes asíncronos son aquellos tal que el emisor puede enviar nuevos mensajes mientras el original está siendo procesado. El mensaje asíncrono ocurre en el tiempo de manera independiente a otros mensajes. Los mensajes síncronos son todo lo contrario, el emisor debe esperar a que termine el tiempo de proceso del mensaje antes de que pueda emitir nuevos mensajes. UML emplea los siguientes convenios para representar el tipo de mensaje.

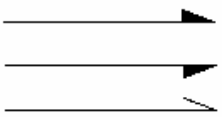
Símbolo	Significado
	Mensaje simple que puede ser síncrono o asíncrono.
	Mensaje simple de vuelta (opcional).
	Mensaje síncrono.
	Mensaje asíncrono.

Tabla 2.2.2 Tipos de mensaje en diagramas de interacción

Los diagramas de colaboración son otro tipo de diagramas de interacción, que contiene la misma información que los de secuencia, sólo que se centran en las

responsabilidades de cada objeto, en lugar de en el tiempo en que los mensajes son enviados. Cada mensaje de un diagrama de colaboración tiene un número de secuencia. El primer nivel de la secuencia es 1, y los mensajes que son enviados durante la misma llamada a un método se numeran 1.1, 1.2 y así sucesivamente para tantos niveles como sea necesario.

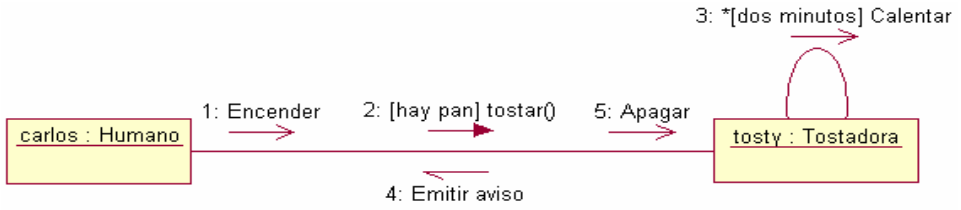


Figura 2.2.9 Diagrama de Colaboración

2.2.10 Diagrama de Estados y Diagrama de Actividades

Los diagramas de estados muestran los posibles estados en que puede encontrarse un objeto y las transiciones que pueden causar un cambio de estado. El estado de un objeto depende de la actividad que esté llevando a cabo o de alguna condición.

Las transiciones son las líneas que unen los diferentes estados. En ellas se representa la condición que provoca el cambio, seguida de la acción oportuna separada por **"/"**.

Los estados inicial, a partir del que se "entra" en la máquina de estados, y final, que indica que la máquina de estados termina, no tienen otro significado adicional, son elementos ornamentales y se representan mediante un circulo negro y un circulo negro resaltado respectivamente.

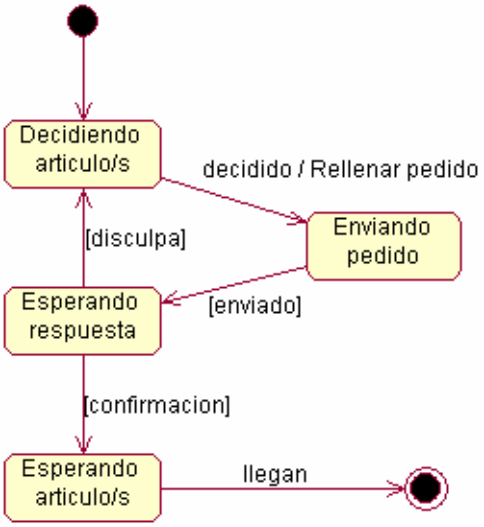


Figura 2.210 Máquina de Estados, estados simples

Los diagramas de actividades son básicamente diagramas de flujo adornados, que guardan mucha similitud con los diagramas de estados. Mientras que los diagramas de estados centran su atención en el proceso que está llevando a cabo un objeto, los diagramas de actividades muestran como las actividades fluyen y las dependencias entre ellas.

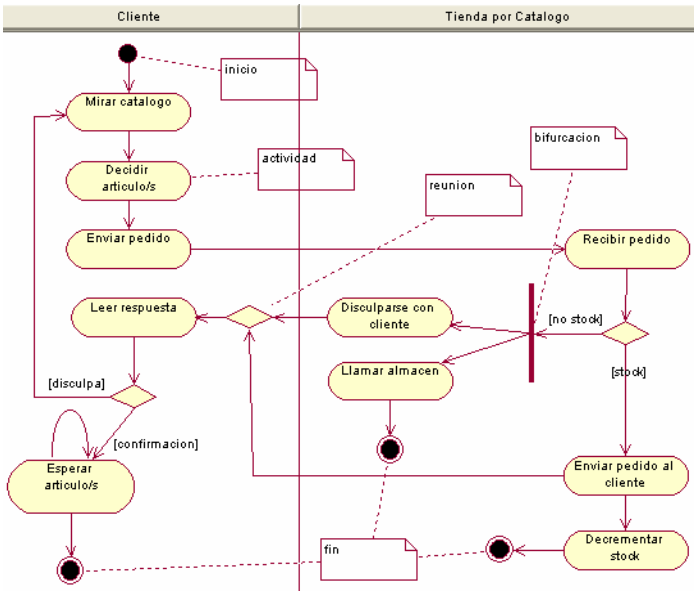


Figura 2.2.12 Diagrama de Actividades

2.3 Herramientas Web

Existen diversas herramientas para desarrollar aplicaciones Web, éstas se dividen en Herramientas Web Comerciales y Open Source.

2.3.1 Comerciales

El software comercial también llamado software propietario, se refiere a cualquier programa informático en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo o que su código fuente no está disponible o el acceso a éste se encuentra restringido.

2.3.1.1 Visual Basic .NET

Visual Basic sigue es una de las herramienta más productivas para la creación de aplicaciones que se ejecutan en el sistema operativo Microsoft Windows. Con Visual Basic .NET, los programadores pueden aprovechar sus conocimientos y capacidades para crear la próxima generación de aplicaciones y servicios Web XML.

Aplicaciones para Windows y el Web
Con la herencia visual, los programadores pueden simplificar enormemente la creación de aplicaciones basadas en Windows, centralizando la interfaz de usuario y la lógica común de toda su solución en formularios primarios. Utilizando delimitadores y acoplamiento de controles, los programadores pueden generar formularios redimensionables automáticamente sin código, mientras el editor de menús in situ permite crear menús de manera visual directamente desde el Diseñador de Windows Forms.
Por el lado del Web, se pueden crear soluciones Web en Visual Basic .NET utilizando el Diseñador de Web Forms y el Diseñador XML compartidos. Los programadores pueden utilizar la tecnología Microsoft IntelliSense y la capacidad para completar etiquetas; o bien, elegir el editor WYSIWYG (lo que ve es lo que se imprime) para poder crear aplicaciones Web interactivas arrastrando y colocando elementos.

Aplicaciones móviles

Mobile Internet de Visual Studio .NET. permite llegar hasta la gama más amplia de dispositivos compatibles con Internet. Estas nuevas características ofrecen a los programadores una única interfaz Web móvil para proporcionar compatibilidad con una amplia gama de dispositivos Web, incluidos WML 1.1 para teléfonos móviles WAP, HTML compacto (cHTML) para teléfonos i-mode y HTML para Pocket PC, dispositivos de mano y localizadores (pagers).

Características

La página de inicio de Visual Basic .NET es un portal para programadores que permite tener acceso con un solo clic a información acerca de los proyectos usados recientemente, las preferencias personales, las actualizaciones de productos y la comunidad MSDN Online. El Cuadro de herramientas ampliado muestra una vista dinámica de los componentes del proyecto, como los controles de Windows Forms y Web Forms, los elementos HTML, los objetos y los miniprogramas.

La plantilla de servicios Web XML crea e implementa automáticamente los diversos componentes de un servicio Web. El Asistente para la instalación permite distribuir las aplicaciones .NET de forma sencilla.

La Ayuda dinámica proporciona acceso con un solo clic a la ayuda pertinente, independientemente de la tarea que se esté realizando. MSDN Online Access proporciona vínculos directos a ejemplos, grupos de noticias, actualizaciones y descargas de Visual Basic .NET en el entorno de desarrollo integrado (IDE).

Este lenguaje de programación es uno de los más fáciles de leer y de escribir. Utiliza una compilación en segundo plano que proporciona información al instante y señala los errores con un subrayado ondulante.

Posee una implementación lado a lado que acaba con los conflictos entre versiones y la herencia permite reutilizar el código de cualquier lenguaje basado en .NET. Contiene un control de excepciones estructurado que proporciona un código de control de errores más elegante y fácil de mantener.

Puede incorporar recursos, componentes y código de más de 3 millones de programadores de Visual Basic de todo el mundo y puede utilizar componentes del gran mercado de proveedores de controles para crear completas aplicaciones basadas en este Visual Basic .NET.

2.3.1.2 ASP

ASP (Active Server Pages) es la tecnología desarrollada por Microsoft para la creación de páginas dinámicas del servidor. ASP, el cual se escribe en la misma página Web, utilizando el lenguaje Visual Basic Script o Jscript (Javascript de Microsoft).

Este es un lenguaje del lado del servidor, ya que se ejecuta en el servidor Web, justo antes de que se envíe la página a través de Internet al cliente. Debido a lo anterior, las páginas creadas en ASP, que se ejecutan en el servidor pueden realizar accesos a bases de datos, conexiones en red, y otras tareas para crear la página final que verá el cliente. El cliente solamente recibe una página con el código HTML resultante de la ejecución de la página ASP. Como la página resultante contiene únicamente código HTML, es compatible con todos los navegadores.

El tipo de servidores que emplean este lenguaje son, evidentemente, todos aquellos que funcionan con sistema Windows NT, aunque también se puede utilizar en una PC con Windows 98 si se instala un servidor denominado Personal Web Server. Incluso en sistemas Linux podemos utilizar las ASP si se instala un componente denominado Chilisoft, aunque es mejor trabajar sobre el servidor web para el que está pensado: Internet Information Server.

Con las ASP se pueden realizar muchos tipos de aplicaciones distintas. Permite acceso a bases de datos, al sistema de archivos del servidor y en general a todos los recursos que tenga el propio servidor. También se tiene la posibilidad de comprar componentes ActiveX fabricados por distintas empresas de desarrollo de software que sirven para realizar múltiples usos, como el envío de correo, generar gráficas dinámicamente, etc.

Actualmente se ha presentado ya la segunda versión de ASP, el ASP.NET, que comprende algunas mejoras en cuanto a posibilidades del lenguaje y rapidez con la que funciona. ASP.NET tiene algunas diferencias en cuanto a sintaxis con el ASP, de modo que se ha de tratar de distinta manera uno de otro.

2.3.2 OpenSource

Software libre es el software que, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente. El software libre suele estar disponible gratuitamente en Internet

2.3.2.1 PHP

PHP es un lenguaje de programación usado generalmente para la creación de contenido para sitios Web. PHP es el (acrónimo recursivo de "PHP: Hypertext Preprocessor", inicialmente PHP Tools, o, *P*ersonal *H*ome *P*age Tools) es un lenguaje interpretado usado para la creación de aplicaciones para servidores, o creación de contenido dinámico para sitios Web, y últimamente también para la creación de otro tipo de programas incluyendo aplicaciones con interfaz gráfica usando la librería GTK+.

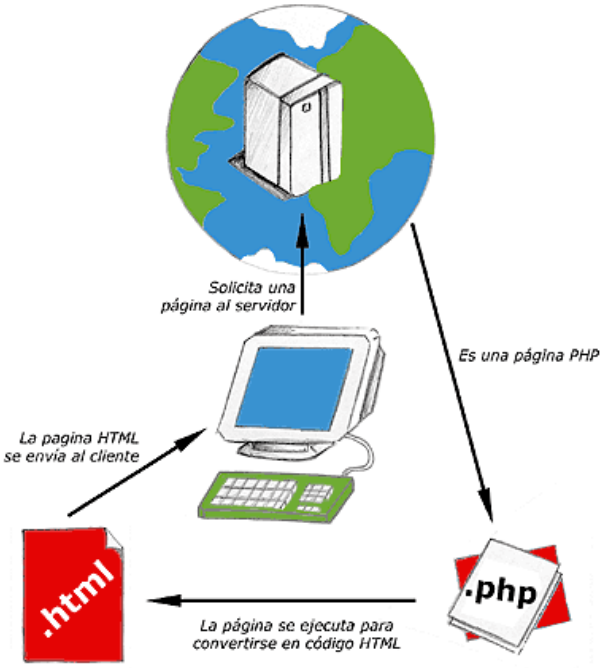


Figura 2.3.2.1 Flujo páginas PHP

Visión general

El fácil uso y la similaridad con los lenguajes más comunes de programación estructurada, como C y Perl, permiten a la mayoría de los programadores experimentados crear aplicaciones complejas con una curva de aprendizaje muy suave. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones y prácticas.

Debido al diseño de PHP, también es posible crear aplicaciones con una interfaz gráfica para el usuario (también llamada GUI), utilizando la extensión PHP-GTK. También puede ser usado desde la línea de órdenes, de la misma manera como Perl o Python puden hacerlo, esta versión de PHP se llama PHP CLI (*Command Line Interface*).

Su interpretación y ejecución se da en el servidor, en el cual se encuentra almacenado el script, y el cliente sólo recibe el resultado de la ejecución. Cuando el cliente hace una petición al servidor para que le envíe una página Web, generada por un script PHP, el servidor ejecuta el intérprete de PHP, el cual

procesa el script solicitado que generará el contenido de manera dinámica, pudiendo modificar el contenido a enviar, y regresa el resultado al servidor, el cual se encarga de regresárselo al cliente. El flujo de información antes mencionado se muestra en la Figura 2.3.1 Flujo páginas PHP. Además es posible utilizar PHP para generar archivos PDF, Flash, así como imágenes en diferentes formatos, entre otras cosas.

Permite la conexión a diferentes tipos de servidores de bases de datos tales como MySQL, Postgres, Oracle, ODBC, IBM DB2, Microsoft SQL Server, Firebird y SQLite; lo cual permite la creación de Aplicaciones web muy robustas.

PHP también tiene la capacidad de ser ejecutado en la mayoría de los sistemas operativos tales como UNIX (y de ese tipo, como Linux), Windows y Mac OS X, y puede interactuar con los servidores de Web más populares ya que existe en versión CGI, módulo para Apache, e ISAPI.

Usos de PHP

Los principales usos del PHP son los siguientes:

Programación de páginas Web dinámicas, habitualmente en combinación con el motor de base datos MySQL, aunque cuenta con soporte nativo para otros motores, incluyendo el estándar ODBC, lo que amplía en gran medida sus posibilidades de conexión.

Programación en consola, al estilo de Perl, en Linux, Windows y Macintosh.

Creación de aplicaciones gráficas independientes del navegador, por medio de la combinación de PHP y GTK (GIMP Tool Kit), que permite desarrollar aplicaciones de escritorio tanto para los sistemas operativos basados en Unix, como para Windows y Mac OS X.

Ventajas de PHP

La principal ventaja se basa en ser un lenguaje multiplataforma.

Capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad.

Leer y manipular datos desde diversas fuentes, incluyendo datos que pueden ingresar los usuarios desde formularios HTML

Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados ext's o extensiones).

Posee una muy buena documentación en su página oficial.

2.3.2.2 Java

Java es un lenguaje de programación orientado a objetos desarrollado por James Gosling y sus compañeros de Sun Microsystems al inicio de la década de 1990. A diferencia de los lenguajes de programación convencionales, que generalmente están diseñados para ser compilados a código nativo, Java es compilado en un bytecode que es ejecutado (usando normalmente un compilador JIT), por una máquina virtual Java.

El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos mucho más simple y elimina herramientas de bajo nivel como punteros.

En el Web cliente

La capacidad de los navegadores Web para ejecutar applets de Java ha asegurado la continuidad del uso de Java por el gran público. Java suele usarse para aplicaciones más complejas como la zona de juegos o reproductores de video.

En el Web servidor

En la parte del servidor, Java es más popular que nunca, con muchos sitios empleando páginas JavaServer, conectores como Tomcat para Apache y otras tecnologías Java.

Existen aplicaciones Java cuyo uso está ampliamente extendido, como los NetBeans, el entorno de desarrollo (IDE) Eclipse, y otros programas como LimeWire y Azureus para intercambio de archivos.

El lenguaje Java se creó con cinco objetivos principales:
Usar la metodología de la programación orientada a objetos.
Permitir la ejecución de un mismo programa en múltiples sistemas operativos.
Incluir por defecto soporte para trabajo en red.
Diseñarse para ejecutar código en sistemas remotos de forma segura.
Fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

2.3.2.3 J2EE

Java 2 (J2EE) es una plataforma que habilita soluciones para desarrollo, uso efectivo y manejo de multicapas en aplicaciones centralizadas en el servidor.
La plataforma Java 2, Enterprise Edition (J2EE) es fruto de la colaboración de SUN con los líderes del sector del software empresarial (IBM, Apple, Bea Systems, Oracle, Inprise, Hewlett-Packard, Novell, etc.) para definir una plataforma robusta y flexible orientada a cubrir las necesidades empresariales en e-business y business-to-business.

J2EE utiliza la plataforma Java 2 SE, para tender una completa, estable, segura, y rápida plataforma Java en el ámbito de la empresa. Permite ahorrar a la compañía, porque habilita una plataforma que reduce de manera significativa los costos y la complejidad de desarrollo de soluciones multicapas, resultando en servicios que pueden ser desarrollados rápidamente y ampliados fácilmente.

Tecnologías que son incluidas en J2EE:

- Enterprise JavaBeansTM
- JavaServers PagesTM
- Servlets Java Naming and Directory InterfaceTM (JNDI)
- Java Transaction API (JTA)
- CORBA
- API de acceso a datos JDBC.

Especificaciones J2EE

Java 2, Enterprise Edition, aprovecha muchas de las características de la plataforma Java, como la portabilidad "Write Once, Run Anywhere", el *Application Program Interface* (API) JDBC para el acceso a bases de datos, las tecnología CORBA para la interacción con los recursos existentes de la empresa y un modelo de seguridad que protege los datos incluso en la aplicaciones para Internet. Sobre esta base, Java 2 Enterprise Edition añade el soporte completo para componentes Enterprise Java Beans, el API Java Servlets y la tecnología JavaServer Pages. El estándar J2EE incluye todas las especificaciones y pruebas de conformidad que permiten la portabilidad de las aplicaciones a través de la amplia gama de sistemas empresariales compatibles con J2EE.

Arquitectura de J2EE

J2EE esta basado en la arquitectura del lado del servidor (Served-based). Este tipo de arquitectura concentra la mayoría de los procesos de la aplicación en el servidor o en un pedazo de este. Este tpo de arquitectura tiene dos ventajas críticas en comparación con los otros tipos, estos son:

Múltiples Clientes: Una arquitectura basada en el servidor requiere una clara separación entre la capa cliente (interfaz) y la capa servidor, en la cual se realizan los procesos de la aplicación. Esto permite que una simple aplicación soporte simultáneamente clientes con distintos tipos de interfaces, incluyendo poderosas interfaces (gráficas) para equipos corporativos, interfaces multimedia interactivas para usuarios con conexiones de alta velocidad, interfaces eficientes basadas en texto para usuarios con conexiones de baja velocidad, etc.

Operaciones robustas: Una arquitectura basada en el servidor soporta escalabilidad, confiabilidad, disponibilidad y recuperabilidad. Aplicaciones basadas en el servidor pueden ser divididas y distribuidas en múltiples procesadores. Componentes de la aplicación pueden ser replicados para dar soporte a caídas instantáneamente.

La plataforma de J2EE provee un conjunto de APIs de java y servicios necesarios para el soporte de aplicaciones para empresas. La plataforma completa puede ser implementada en un solo sistema, o la plataforma de servicios puede ser distribuida a través de varios sistemas, pero todas las APIs especificadas deben ser incluidas en alguna parte del sistema completo.

Los desarrolladores necesitan J2EE porque escribir aplicaciones distribuidas en empresas es muy duro, y ellos necesitan una solución de alta productividad que les permita enfocarse sólo en escribir la lógica y tener un completo rango de servicios en que confiar (enterprise-class), como objetos de transacciones distribuidas o middleware orientado a objetos, etc.

2.3.2.4 CGI (Common Gateway Interface).

Sistema para ampliar la funcionalidad básica de un servidor Web. Ejecutando programas o scripts en el servidor, como respuestas a peticiones realizadas por el servidor.
La forma lógica es utilizar CGI en formularios HTML, donde el navegador envía el formulario a un script CGI en el servidor y este devuelve resultados. Por ejemplo, para copiar datos de un formulario a un correo y enviarlo, o para guardar información en una base de datos, (se suelen programar en C++ o PERL, e incluso en VBScript en el servidor).

2.3.2.5 Perl

Es un lenguaje de programación muy utilizado para construir aplicaciones CGI para el Web. Perl es un acrónimo de Practical Extracting and Reporting Languaje, que viene a indicar que se trata de un lenguaje de programación muy práctico para extraer información de archivos de texto y generar informes a partir del contendido de los ficheros.
Perl es un lenguaje de programación interpretado, al igual que muchos otros lenguajes de Internet como Javascript o ASP. Esto quiere decir que el código de los scripts en Perl no se compila sino que cada vez que se quiere ejecutar se lee

el código y se pone en marcha interpretando lo que hay escrito. Además es extensible a partir de otros lenguajes, ya que desde Perl podremos hacer llamadas a subprogramas escritos en otros lenguajes. También desde otros lenguajes podremos ejecutar código Perl.
Perl está inspirado a partir de lenguajes como C, sh, awk y sed (algunos provenientes de los sistemas Uníx), pero está enfocado a ser más práctico y fácil que estos últimos. Es por ello que un programador que haya trabajado con el lenguaje C y los otros tendrá menos problemas en entenderlo y utilizarlo rápidamente. Una diferencia fundamental de Perl con respecto a los otros lenguajes es que no limita el tamaño de los datos con los que trabaja, el límite lo pone la memoria que e n ese momento se encuentre disponible.
Si se quiere trabajar con Perl será necesario tener instalado el interprete del lenguaje. A partir de ese momento podemos ejecutar CGIs en nuestros servidores web.

2.4 Patrones de diseño

Un patrón de diseño es una solución recurrente a un problema en un entorno, situación o bajo ciertas condiciones interrelacionadas. Los diseñadores de software basándose en la experiencia han desarrollado la capacidad de identificar los patrones que caracterizan un problema y con ello documentar la solución para casos similares, por lo que los patrones de diseño nos permiten la capacidad de reutilización de código.
Los patrones del diseño tratan los problemas del diseño que se repiten y que se presentan en situaciones particulares del diseño, con el fin de proponer soluciones a ellas. Por lo tanto, los patrones de diseño son soluciones exitosas a problemas comunes. Existen muchas formas de implementar patrones de diseño. Los detalles de las implementaciones son llamadas estrategias.

Las características que debe de tener un patrón de diseño son las siguientes:
Nombre: Describe el problema de diseño, su solución, y consecuencias en una o dos palabras. Tener un vocabulario de patrones nos permite hablar sobre ellos

Problema: Describe cuando aplicar el patrón. Se explica el problema y su contexto. Puede describir estructuras de clases u objetos que son sintomáticas de un diseño inflexible. Se incluye una lista de condiciones

Solución: Describe los elementos que forma el diseño, sus relaciones, responsabilidades y colaboraciones. No se describe un diseño particular. Un patrón es una plantilla

Consecuencias: Son los resultados de aplicar el patrón

Existen tres categorías o tipos de patrones de diseño, los cuales son:

Creacionales: Tratan la forma de crear instancias de las clases. Su objetivo es ocultar los detalles de cómo son creados los objetos con la finalidad de facilitar su manejo. Ejemplos de este patrón tenemos al Abstract Factory, Builder, Singleton, Prototype.

Estructurales: Describen la forma de cómo combinar objetos y clases para la creación de clases o componentes más complejos. Ejemplos de estos patrones son: Adapter, Bridge, Composite, Decorador, Fachada, Proxy, entre otros.

Comportamiento: Definen la comunicación e iteración entre objetos de un sistema con la finalidad de reducir el acoplamiento entre objetos. Ejemplos: Chain of Responsability, Command, Interpreter, Iterator, Mediator, Memento, Observer, Visitor, entre otros.

2.4.1 Catálogo de Patrones

Los patrones de diseño se han ido documentando y se han ido agrupando en catálogos con la finalidad de hacerlos útiles y accesibles para el desarrollo del software en el momento en que éstos se requieran. El catálogo de patrones más famoso es el contenido en el libro "Design Patterns: Elements of Reusable Object-Oriented Software" también conocido como el libro GoF (Gang-Of-Four Book), del cual mencionaremos algunos de los más utilizados a continuación:

2.4.1.1 Abstract Factory

Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar su clase concreta. Se compone de los siguientes elementos:

AbstractFactory: Declara una interfaz de operaciones que crean productos abstractos.

ConcreteFactory: Implementa las operaciones que crean los objetos producto

AbstractProduct: Declara una interfaz para un tipo de objeto producto

Product: Define un objeto producto que será creado por el correspondiente ConcreteFactory

Client: Usa las interfaces

En la figura 2.4.1 se muestra el diagrama de clases de este patrón

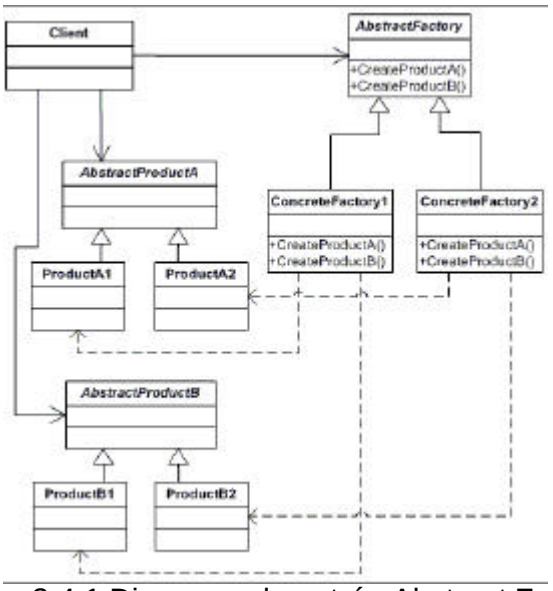


Figura 2.4.1 Diagrama de patrón Abstract Factory

2.4.1.2 Factory Method

Define una interfaz para crear un objeto dejando a las subclases decidir el tipo específico al que pertenecen. Se compone de los siguientes elementos:

Creator: Declara el método FactoryMethod(). Se puede definir una implementación por defecto para él

ConcreteCreator: Sobrescribe el método FactoryMethod(), devolviendo una instancia de ConcreteProduct

Product: Define la interfaz de objetos que crea el método FactoryMethod

ConcreteProduct: Implementa la interfaz del producto

En la figura 2.4.2 se muestra el diagrama de clases del patrón Factory Method.

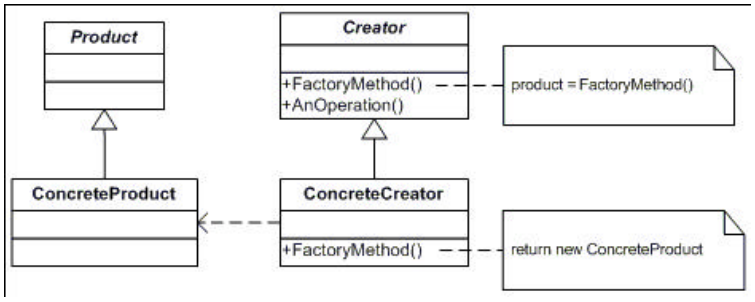


Figura 2.4.2 Diagrama de patrón Factory Method

2.4.1.3 Singleton

Garantiza que solamente se crea una instancia de la clase y provee un punto de acceso global a él. Define un método Instance() que permite al cliente el acceso a su única instancia. El método Instance() es estático. Esta misma clase es la responsable de la creación y mantenimiento de su propia instancia

2.4.1.4 Adapter

Convierte la interfaz que ofrece una clase en otra esperada por los clientes. Sus elementos son los siguientes:

Target: Define una interfaz de dominio específico que el cliente utiliza

Adapter Adapta interfaces de Adaptee y Target

Adaptee: Define una interfaz que existiendo necesita adaptación

Client: utiliza objetos ajustándose a la interfaz ofrecida por Target

2.4.1.5 Decorador

Extiende la funcionalidad de un objeto dinámicamente de tal modo que es transparente a sus clientes. Se compone de los siguientes elementos:

Component Define la interfaz que ofrecen los objetos que poseen Responsabilidades añadidas dinámicamente

ConcreteComponent: Define un objeto al que responsabilidades adicionales pueden serle añadidas

Decorator: Mantiene referencia a un objeto Component y define una interfaz que conforma la interfaz del Component

ConcreteDecorator: Añade las responsabilidades al Component

2.4.1.6 Fachada (Façade)

Simplifica los accesos a un conjunto de objetos relacionados proporcionando un objeto de comunicación. Se compone de los siguientes elementos:

Façade: Conoce que clases son responsables de que peticiones y así, delega las peticiones a los objetos apropiados

Subsystem: Implementa la funcionalidad del subsistema, realiza el trabajo solicitado por el objeto Façade y no conoce, ni mantiene referencia alguna del objeto Façade

En la figura 2.4.3 se muestra el diagrama de clases del patrón Fachada

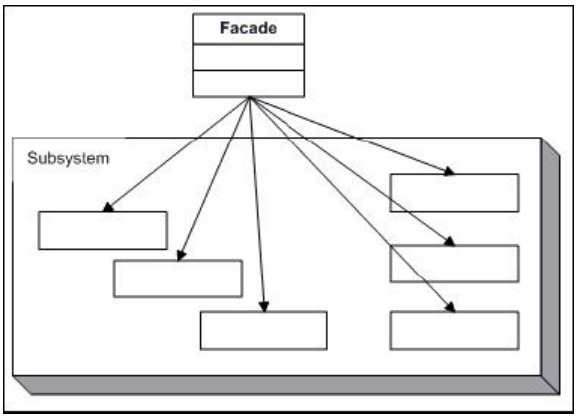


Figura 2.4.3 Diagrama de patrón Fachada

2.4.1.7 Command

Encapsula una petición de un comando como un objeto. Se compone de las siguientes clases:

Command: Declara una interface para la ejecución de una operación

ConcreteCommand: Define un vínculo entre un objeto Receiver y una acción, implementa Execute() invocando al método correspondiente de Receiver

Cliente: Crea un objeto ConcreteCommand y establece su receptor

Invoker: Pide a **Command** que lleve a cabo su petición

Receiver: Sabe cómo realizar las operaciones asociadas con la puesta en marcha de la petición.

2.4.1.8 Iterator

Permite el acceso secuencial a los elementos de una colección. Se compone de los siguientes elementos:

Iterator Define una interfaz para atravesar y acceder a los elementos.

ConcreteIterator: Implementa la interfaz del iterator, mantiene un puntero al conjunto de elementos.

Aggregate: Define un interfaz para crear un objeto iterator

ConcreteAggregator: Implementa la interfaz de creación del Iterator devolviendo un objeto de ConcreteIterator apropiado.

En la figura 2.4.4 se muestra el diagrama de clases del patrón Iterator

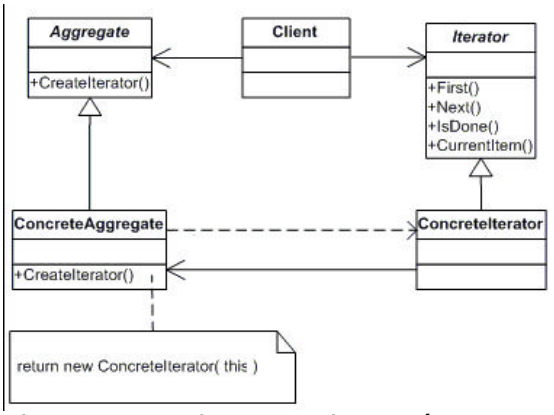


Figura 2.4.4 Diagrama de patrón Iterator

2.4.1.9 Observer

Una forma de notificar cambios a diferentes clases dependientes. Se compone de los siguientes elementos:

Subject Conoce sus observadores y proporciona una interfaz para gestionarlos.

ConcreteSubject: Almacena el estado de interés y envía una notificación a sus observadores cuando su estado cambia.

Observer: Define una interfaz para los objetos a los que debe notificarse el cambio de estado en ConcreteSubject

ConcreteObserver: Mantiene una referencia a un objeto ConcreteSubject, mantiene información consistente relacionada con el estado implementando el método Update().

En la figura 2.4.5 se muestra el diagrama de clases del patrón Observer

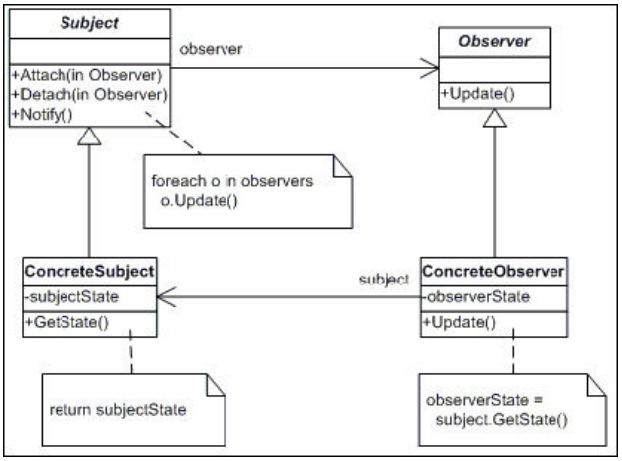


Figura 2.4.5 Diagrama de patrón Observer

2.4.1.10 State

Modifica el comportamiento de un objeto cuando su estado interno cambia. Se compone de los siguientes elementos:

Context: Define la interfaz de interés a clientes.

State: Define una interfaz para encapsular el comportamiento asociado con un estado particular del contexto.

ConcreteState: Cada subclase implementa un comportamiento asociado con un estado de Context.

2.4.1.11Strategy

Define una familia de algoritmos, encapsula cada uno y los hace intercambiables. Se compone de los siguientes elementos

Strategy: Declara una interface común para dar soporte a todos los algoritmos Context utiliza esta interfaz para llamar al algoritmo definido por un ConcreteStrategy

ConcreteStrategy: Implementa el algoritmo usando la interfaz Strategy

Context: De configura con un objeto ConcreteStrategy, sobre el que mantiene una referencia, puede definir una interfaz que permita a Strategy acceder a sus datos

2.4.1.12 Template Method

Define un esqueleto de algoritmo y delega partes concretas de un algoritmo a las subclases. Se compone de los siguientes elementos:

AbstractClass: Define operaciones primitivas abstractas cuya conc reción se delega a las subclases, estas primitivas se utilizan en el cuerpo de un algoritmo esqueleto

ConcreteClass: Implementa las operaciones primitivas abstractas anteriores

2.4.1.13 Visitor

Representa una operación que será realizada sobre los elementos de una estructura de objetos, permitiendo definir nuevas operaciones sin cambiar las clases de los elementos sobre los que opera. Se compone de los siguientes elementos:

Visitor: Establece las operaciones a realizar.

ConcreteVisitor: Implementa dichas operaciones

Element: Define un método Accept() que toma un Visitor como argumento.

ConcreteElement: Implementa el método Accept().

2.4.2 MVC (Model - View - Controller)

Para el diseño de aplicaciones con sofisticados interfaces se utiliza el patrón de diseño Model View Controller o bien Modelo Vista Controlador (MVC) que consiste en separar los datos de una aplicación, la interfaz de usuario, y la lógica de control. 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. Los tres componentes del MVC sol los siguientes:

Modelo (Model): Esta es la representación específica del dominio de la información sobre la cual funciona la aplicación. El modelo es otra forma de llamar a la capa de negocio. La lógica de negocio añade significado a los datos, definiendo las reglas del mismo; por ejemplo, el cálculo del total de un carrito de compras.

Vista (View): Este presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de interfaz de usuario, por ejemplo las páginas JSP y elementos HTML, un botón, etc.

Controlador (Controller): Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Muchas aplicaciones utilizan un mecanismo de almacenamiento persistente (como puede ser una base de datos) para almacenar los datos. MVC no menciona específicamente esta capa de acceso a datos. En la figura 2.4.6 se muestra el diagrama de colaboración del patrón MVC donde se muestran cada una de las interacciones entre los componentes.

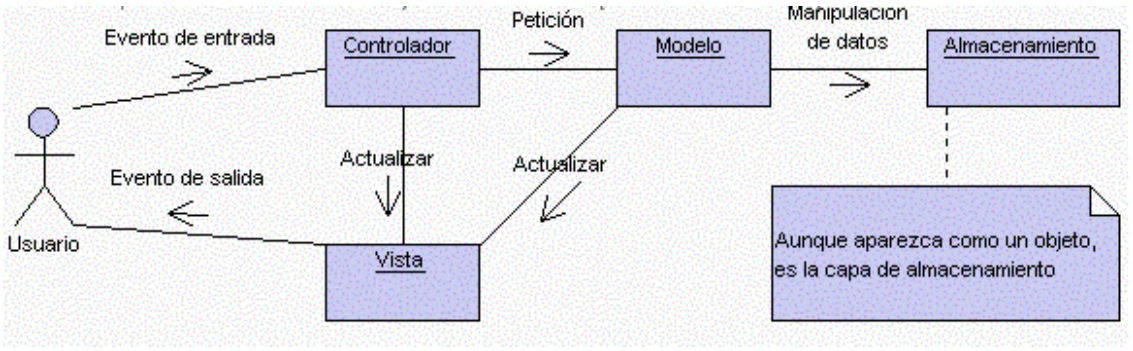


Figura 2.4.6 Diagrama de colaboración del MVC

Es común pensar que una aplicación tiene tres capas principales: presentación (IU), dominio, y acceso a datos. En MVC, la capa de presentación está partida en controlador y vista. La principal separación es entre presentación y dominio; la separación entre V/C es menos clara.

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

- El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace).
- 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.
- 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). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
- 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. 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.

En la figura 2.4.7 se muestra el diagrama de secuencia del patrón MVC donde se muestra el orden de funcionamiento entre componentes.

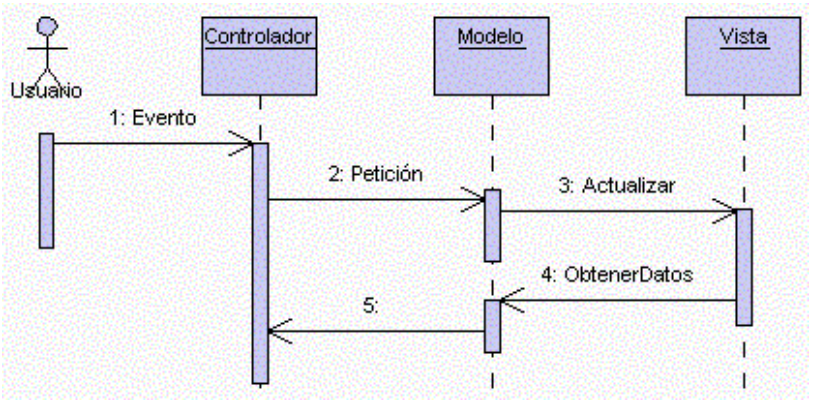


Figura 2.4.7 Diagrama de secuencia del MVC

A su vez el modelo MVC nos permite hacer el diseño de un framework que es una arquitectura de software con componentes bien definidos y reutilizables en la cual otro proyecto de software puede ser organizado y desarrollado de una manera rápida y eficiente, además de definir una metodología de trabajo entre el equipo de desarrollo de software.

2.5 Framework

Los frameworks son una parte importante de la moderna ingeniería del software. El desarrollo del framework está ganando rápidamente la aceptación debido a su capacidad para promover la reutilización del código del diseño y el código fuente.

2.5.1 Definición

Un framework es un conjunto de clases que definen un diseño abstracto para solucionar un conjunto de problemas relacionados. Esta estructura tiene la capacidad de reusabilidad de código, lo que permite una productividad mayor y un tiempo menor en el desarrollo de aplicaciones, en comparación con el desarrollo tradicional de los sistemas de software.

Un framework provee una metodología de trabajo que tiene como objetivo generar aplicaciones de un dominio entero. Por lo tanto, deben existir puntos de flexibilidad que consisten en la modificación de los requisitos particulares para ajustarse a la

aplicación. Los puntos flexibles de un framework se les llama hot-spots (puntos calientes) que son las clases o los métodos abstractos que deber ser implementados o puestos en ejecución. Para generar un ejecutable se debe instanciar el framework (entendiendo como instanciación, la acción de crear un objeto dando valores a las variables en una clase).

Algunas de las características del framework no son mutables ni tampoco pueden ser alteradas fácilmente. Los puntos inmutables o también llamados frozen-spots (puntos congelados) constituyen el núcleo o Kernel de un framework, y son una pequeña parte de código puesto en ejecución que llaman a uno o más puntos flexibles proporcionados por el desarrollador de la aplicación.

2.5.2 Desarrollo de un Framework

Las tres etapas principales del desarrollo del framework son análisis del dominio, diseño del framework, y la instanciación del framework.

En el análisis se plantean los requisitos generales del dominio y los requerimientos que se tendrán en el futuro. Para establecer los requerimientos es conveniente basarse en sistemas de software similares, en experiencias propias de los desarrolladores y en estándares existentes.
En la fase del diseño del framework se define las abstracciones de éste. Se modelan los puntos flexibles y los puntos inmutables (puede ser con diagramas de UML, Modelo Unificado del Lenguaje). En esta etapa se utilizan los patrones de diseño.

En la fase de instanciación los hot-spots del framework son implementados, generando un software del sistema. Cabe mencionar que cada una estas aplicaciones tendrán los frozen-spots del framework en común. En la Figura 2.5.1 se muestra las fases de desarrollo del framework.

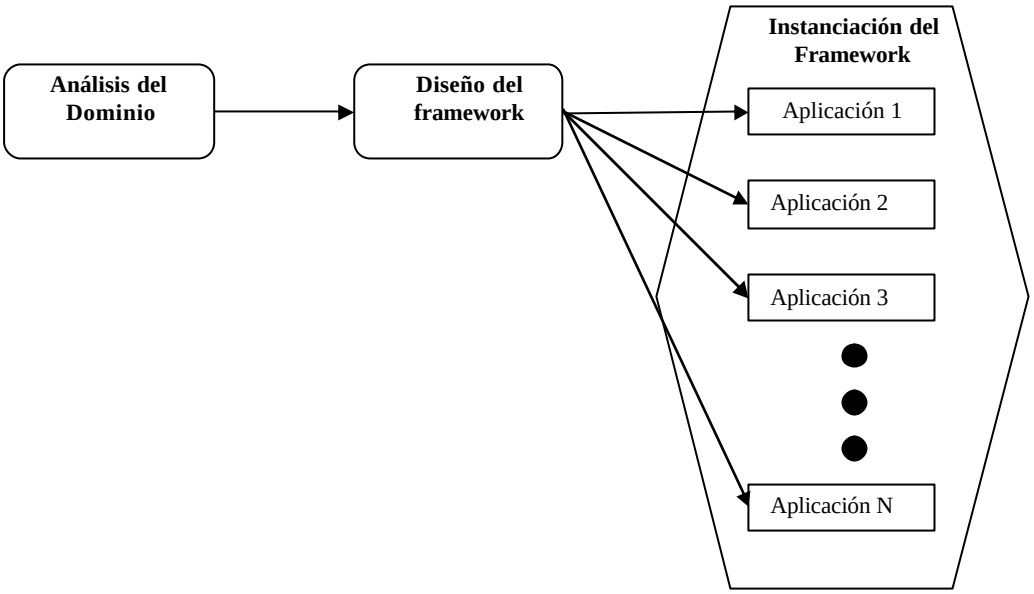


Fig. 2.5.1 Proceso de desarrollo del Framework

En el desarrollo del framework se obtienen requisitos para un dominio entero y varias aplicaciones pueden resultar a partir de la fase de instanciación.

Existen frameworks donde la instanciación solamente es posible a través de la creación de nuevas clases. Estas clases y el código se pueden introducir en el marco por herencia o composición. Se debe programar el framework y entenderlo muy bien para producir un caso.

Hay otro tipo de frameworks que producen instancias usando scripts de configuración. Después de la configuración, una herramienta automática de instanciación crea las clases y el código de fuente. Estos frameworks son generalmente más fáciles de usar.

Durante el desarrollo del framework se deben considerar los siguientes aspectos para lograr una alta funcionalidad de la estructura:

Flexibilidad. Los frameworks se construyen para dar flexibilidad y generalidad, tratando de cubrir un dominio entero y no problemas determinados. Los

desarrollos del funcionamiento junto con el uso abusivo de puntos flexibles en un diseño del framework conducen a sistemas de software complejos. Por esto, es importante contemplar un equilibrio entre flexibilidad y funcionalidad, para elegir los puntos flexibles cuidadosamente, sin crear un framework que sea demasiado genérico.

Integración. En ocasiones es necesario integrar frameworks para satisfacer los requerimientos de una aplicación, pero no siempre resulta fácil lograr la integración. Los problemas más comunes que los desarrolladores de frameworks y aplicaciones encuentran al integrar dos o más frameworks se derivan de un conjunto de cinco causas comunes: comportamiento cohesivo, cobertura del dominio, intención del diseño, carencia de acceso al código fuente y carencia de los estándares para el framework.

Documentación. La documentación debe hacer referencia a las formas de hacer fácil y eficiente la modificación y adición de funcionalidad además de tener herramientas prácticas de instanciación. Las guías de consultas y documentación pueden ser:

- Tarjetas que se enfoquen a los puntos flexibles del framework.
- Manuales que expliquen cómo el framework debe ser puesto en ejecución y los pasos requeridos para hacerlo. Los manuales contienen los pasos a seguir que describen cómo solucionar problemas específicos mientras el desarrollador de la aplicación está instanciando el framework.
- La documentación de la configuración, que se utiliza asociando las soluciones arquitectónicas usadas a través del diseño del framework.
- Los patrones de diseño son bien conocidos y también pueden ayudar a la comprensión del proceso de instanciación del framework.

Costo de análisis del dominio. El análisis del dominio intenta caracterizar el tamaño y la complejidad de un dominio elegido. Si el dominio es demasiado grande, el tiempo para desarrollar el framework y el costo del mismo serían

excesivos. Por otra parte, si se elige un dominio demasiado estrecho, se reduce la aplicabilidad del framework, las aplicaciones generadas serán demasiado similares y las razones para construir el framework serán poco justificables. Es importante identificar claramente; los puntos flexibles que son necesarios, cuáles son los que realmente se necesitan en los requerimientos y cuáles son innecesarios o superfluos.

Para que un framework sea altamente aceptable es necesario que las aplicaciones generadas no se vean afectadas al momento de modificar o añadir funcionalidad, es por esto que el framework debe ser bien diseñado e implementado, es decir, debe tener documentación sólida y resolver los requerimientos del dominio. La evaluación del diseño del framework y/o su implementación no siempre es directa; se debe considerar la experiencia del diseñador, la complejidad del proceso del instanciación, los requisitos resueltos y los no resueltos, y también la guía de trabajo, que contiene los planes para poner al día el framework, si es que fuera necesario un rediseño completo en una nueva versión o simplemente una subsiguiente actualización compatible.

2.5.3 Frameworks Existentes para Aplicaciones Web

Estos son algunos ejemplos de frameworks para aplicaciones Web y sus características generales.

Catalyst: Es un framework para aplicaciones Web de código abierto escrito en Perl, soporta la arquitectura MVC, así como también soporta algunos patrones Web experimentales. Está altamente inspirado en Ruby on Rails. Catalyst promueve la re-utilización de los módulos de Perl que soportan bien lo que requieren las páginas Web.

Ruby on Rails: Es un framework de aplicaciones Web de código abierto escrito en el lenguaje de programación Ruby, sigue el paradigma de la arquitectura

Modelo-Vista-Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros frameworks y con un mínimo de configuración. El lenguaje de programación Ruby permite la metaprogramación, de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible.

Jakarta Struts: Es un framework para el desarrollo de aplicaciones Web con arquitectura MVC bajo la plataforma J2EE (Java 2, Enterprise Edition). Struts permite reducir el tiempo de desarrollo, su carácter de software libre y su compatibilidad con todas las plataformas, lo convierte en una herramienta altamente disponible. Struts permite que el desarrollador se concentre en el diseño de aplicaciones complejas como una serie simple de componentes del Modelo y de la Vista intercomunicados por un control centralizado**.**

.NET Framework: Constituye la base de la plataforma .Net y denota la infraestructura sobre la cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en entorno de ejecución distribuido. Los principales componentes del marco de trabajo son: el conjunto de lenguajes de programación, la Biblioteca de Clases Base o BCL, el Entorno Común de Ejecución para Lenguajes o CLR.

Prado: Es un framework para desarrollo rápido de programas Web en PHP 5. Prado reconceptualiza el desarrollo de las aplicaciones Web en términos de componentes, eventos y propiedades, en vez de procedimientos, URLs y parámetros de query.

3 Alternativas de Solución

Para dar una solución en el sentido planteado se puede optar por seguir alguna de las siguientes opciones:

- Adquisición de un framework comercial.
- Uso de un framework de código abierto.
- Construcción de un framework.

Cada una de estas opciones es viable y puede cubrir al 100% los requerimientos planteados, sin embargo, entre ellas existen diferencias que se traducen en ventajas y desventajas en comparación contra las otras, a continuación describimos estas soluciones de forma teórica y sus ventajas y desventajas.

A su vez se hace un análisis de herramientas de programación orientada a objetos que son opciones viables para la construcción del framework, así como las ventajas y desventajas que nos ofrecen cada una de ellas.

3.1 Análisis de herramientas de POO

Para el diseño del framework a diseñar se considerarán las siguientes tres herramientas de programación orientada a objetos debido a que son las más utilizadas en la actualidad para sistemas en un entorno WEB, las cuales son:

- Java
- PHP
- ASP .NET

De las cuales se mencionarán sus ventajas y sus desventajas de cada una de ellas.

3.1.1 Ventajas

Aquí se listan las ventajas que nos ofrecen las herramientas orientadas a objetos seleccionadas, las cuales son las siguientes:

Java

- Es un lenguaje orientado a objetos fuerte, debido a que maneja todos los conceptos necesarios de POO para contar con un diseño de una arquitectura fuerte y robusta.
- Soporta concurrencia. Podemos tener varios hilos de ejecución que, además pueden interactuar entre ellos.
- Se tiene una mayor flexibilidad y facilidad con el manejo de errores y excepciones en las aplicaciones que otros lenguajes.
- Es capaz de interactuar con la gran mayoría de los manejadores de bases de datos existentes de manera muy eficaz y sencilla.
- Interfaces. Java permite la declaración de interfaces, con lo que podemos "emular" la herencia múltiple. Este tipo de clases sirven para declarar los métodos que una clase deberá implementar.
- Soporte para aplicaciones distribuidas. Aporta mecanismos para llamadas a métodos de clases remotas.
- Tiene una aplicación (javadoc) que ayuda a la creación de una documentación completa a nivel código de las clases que estén desarrollando.
- Las aplicaciones que se consiguen son muy seguras, ya que la máquina virtual controla automáticamente los accesos a los recursos, permisos, memoria.
- Las aplicaciones desarrolladas son independientes del sistema operativo y portables debido al concepto de byte code y máquina virtual de Java (Java Virtual Machina JVM).
- Hay varias plataformas según el tipo de aplicación a desarrollar:
- J2SE: La base de todo, para aplicaciones sencillas o distribuidas.
- J2EE: Sistemas Web distribuidos con JSPs, servlets, EJBs (Enterprise Java Beans). Aplicaciones profesionales a nivel empresarial (pero también se puede usar para cosas sencillas).
- J2ME: Para dispositivos de recursos escasos, como PDAs, móviles, etc.
- La documentación existente es abundante y completa.

- Hay extensiones y librerías en gran cantidad.
- La plataforma Java2, Edición Empresarial (J2EE) reduce el costo y complejidad de desarrollo de servicios que ofrecen alta disponibilidad, seguridad, fiabilidad y escalabilidad, y da por resultado servicios que pueden ser creados rápidamente y fácilmente mejorados respondiendo a las demandas competitivas que se presentan en la actualidad.
- Para J2EE existen tanto contenedores de Servlets y JSPs Open Source como Tomcat y Eclipse así como sofisticados servidores de aplicaciones propietarios como Websphere, Oracle IAS, Weblogic, entre otros.

PHP

- Es una herramienta totalmente de código abierto, lo cual se puede hacer su uso sin necesidad de licencias.
- Es fácil de aprender y de utilizar por parte del equipo de desarrollo debido a su sencillez de sintaxis y reglas que son fáciles debido a que son basadas en lenguajes como PERL y C.
- Funciona en servidores WEB de código abierto como Apache y en plataformas Windows con IIS (Internet Information Services).
- Tiene compatibilidad con objetos desde la versión 4.0 y fue mejorada en la versión 5.0
- Existen múltiples librerías de código fuente libres de utilizar.
- Maneja librerías específicas para múltiples manejadores de bases de datos.

ASP .NET

- Es fácil de aprender y utilizar ya que existe la flexibilidad de trabajar con lenguajes de scripting como VBScript, JScript, Perlscript y Pitón, así como lenguajes compilados como VB (Visual Basic), C, C#, Cobol, Smalltalk y Lisp.
- Es orientado a objetos, debido a que soporta la herencia, polimorfismo y encapsulamiento.

- Las librerías de .NET están organizadas en clases heredadas con una funcionalidad bien definida.

3.1.2 Desventajas

Aquí se listan las desventajas que tienen las herramientas orientadas a objetos seleccionadas, las cuales son las siguientes:

Java

- Herencia simple. Las clases Java sólo pueden heredar de otra clase, mientras que otros lenguajes permiten herencia múltiple. (Se cubre esta desventaja con las interfaces).
- Es un lenguaje interpretado. Esto quiere decir que la Máquina Virtual de Java interpreta el byte code (código Java compilado) instrucción a instrucción, lo procesa, y es entonces cuando hace comprobaciones de tipos. Ésto lleva su tiempo, por lo que Java es más lento que otros lenguajes.
- El concepto de clases y objetos es distinto del de lenguajes como C++, y la curva de aprendizaje al principio es más grande (si bien la sintaxis del lenguaje recuerda bastante a C).

PHP

- No es un lenguaje puramente orientado a objetos, debido a que está característica se añadió a partir de la versión 4.0, la cual es menos eficiente que las que ofrecen otros lenguajes orientados a objetos.
- La cantidad de APIs y documentación son limitadas y en ocasiones incompleta.
- Sólo funciona en entornos WEB y en limitados servidores WEB, comúnmente open source, por lo cual el soporte y garantía son limitados,
- El mantenimiento de las aplicaciones desarrolladas es difícil debido a que es un lenguaje con mínimas restricciones sintácticas.
- Escalabilidad limitada
- El manejo de errores y excepciones fue una adición agregada desde la versión 5.0 que en las versiones anteriores no lo contienen.

- Las librerías de conexión a base de datos son específicas para cada manejador, lo cual quita flexibilidad de que al cambiar el manejador de base de datos en una aplicación desarrollada se tenga que cambiar forzosamente código fuente.

ASP .NET

- Sólo funciona en plataformas Microsoft, por lo cual no es portable a otros sistemas operativos
- Es costosa la infraestructura tecnológica y software de desarrollo.
- Su rendimiento consume excesivamente recursos de memoria y de ejecución cuando existe mucha concurrencia de usuarios WEB
- Para conectarse a la base de datos es necesario usar ODBC, lo cual disminuye rendimiento de acceso a las mismas.
- Funciona solo en servidores WEB IIS, los cuales comúnmente tienen huecos de seguridad y son propensos a virus y ataques.

3.2 Adquisición de un framework comercial.

En el mercado actualmente existen productos generalmente asociados a los Servidores de Aplicaciones que solucionan la problemática planteada, esto lo realizan por medio de componentes y herramientas propietarias para el uso de los desarrolladores de sistemas informáticos.

3.2.1 Ventajas.

- La principal ventaja radica en un tiempo de implementación muy corto, generalmente menos de 10 días.
- La integración con el servidor de aplicaciones es alta, por lo que lograr funcionalidad compleja es relativamente sencillo.
- Incluyen generalmente herramientas gráficas para el apoyo al diseño e implementación de componentes.

- Ofrece una amplia gama de opciones de implementación para cubrir casi cualquier tipo de necesidad.
- Incluye soporte y asesoría técnica por parte del proveedor.
- Este tipo de infraestructura ha sido probada ya en varios sistemas informáticos alrededor del mundo, por lo que encontrar un error en los mismos es poco probable, aunque no imposible.
- Generalmente vienen acompañados de más funcionalidad, como pueden ser flujos de trabajo, seguridad, integración con otros sistemas, etc.
- Aunque limitada existen pólizas de garantía para estos productos.

3.2.2 Desventajas.

- El costo de estas herramientas es alto, puede partir desde los 10,000 dólares, y puede ser tan alto como cantidad de servidores y/o desarrolladores se incluyan en la implementación.
- La alta integración con el Servidor de Aplicaciones particular de alguna empresa, dificulta el proceso de migración de componentes a un Servidor de Aplicaciones de otra empresa, y en general habrá que rehacer los componentes para lograr una migración.
- El costo de capacitación y certificación para los desarrolladores es alto, como mínimo 1,000 dólares por curso.
- Si bien el tiempo de implementación es rápido, vencer la curva de aprendizaje lleva el mismo ritmo, y a menudo se requiere la especialización de cierto número de desarrolladores para el manejo de este tipo de framework.
- Queda aún abierta la posibilidad de caer en errores de implementación por parte de los desarrolladores.

En conclusión esta opción es la adecuada para aquellas organizaciones donde exista suficiencia presupuestaria para adquirir estos productos, además que estén dispuestos a comprometerse con un solo Servidor de Aplicaciones y, en su caso,

estar dispuestos a pagar el costo de una migración. Además si no es el único producto que se usará del proveedor en cuestión, puede haber una reducción de costos por compra de grupos de productos.

3.3 Uso de un framework de código abierto.

Existen también iniciativas bien conocidas en el ambiente informático llamadas de código abierto (open source), los cuales son proyectos en los que colabora un grupo de personas, en las que cada una aporta su experiencia para contribuir a este proyecto. Una de las iniciativas mejor conocidas es la del proyecto Struts de Jakarta. Este tipo de iniciativas son una buena alternativa de solución a la problemática planteada.

3.3.1 Ventajas.

- El costo de implementación es mínimo.
- El código fuente es público, lo potencialmente puede significar el control absoluto de la solución.
- Existen foros de desarrolladores, donde se pueden consultar dudas o problemas, además de proponer temas.
- La documentación para su uso es pública.
- Para el caso de componentes Java (J2EE) que utilizan componentes genéricos pueden correr en cualquier servidor de aplicaciones por lo que facilita su portabilidad y migración.

3.3.2 Desventajas.

- La documentación para su uso suele ser en ocasiones magra y no siempre está concentrada en una sola fuente de información.
- Si bien el código fuente es público, no existe documentación o capacitación para su entendimiento, por lo que se requiere de personal que tenga un

gran grado de experiencia en la arquitectura para poder modificar algún componente de manera exitosa.

- No existe soporte o asesoría técnica formal por parte de los que implementaron este framework.
- El margen para realizar una mala implementación es muy amplio.
- Generalmente este tipo de iniciativas con el tiempo pueden caer en desuso, o absorbidas por alguna empresa convirtiéndose entonces en una opción comercial.
- No existe garantía de ningún tipo.
- No cuentan con herramientas gráficas que apoyen la implementación de sistemas, sin embargo, en algunos casos compañías terceras venden productos para este fin.

En conclusión esta opción es la adecuada para aquellas organizaciones donde exista un amplio grado de experiencia y conocimiento en la arquitectura seleccionada. A menudo este tipo de iniciativas pueden ser adoptadas temporalmente en lo que se evalúa una opción comercial.

3.4 Construcción de un framework.

Esta opción es tal vez la más socorrida, sin embargo, a menudo este tipo de framework se integra demasiado con el sistema informático en particular que se está desarrollando por lo que a veces es imposible reutilizarlo para otra iniciativa lo cuál elimina la posibilidad de contemplarlo con una infraestructura. Es por eso que aquí se evalúa a la construcción de una infraestructura reutilizable en más de un sistema y en más de una organización.

3.4.1 Ventajas.

- Se tiene el control completo del funcionamiento de la misma.
- No existe curva de aprendizaje porque es inherente a la construcción del framework.

- Se cubren de manera específica y eficiente requerimientos particulares de cada sistema informático.
- Se puede integrar de forma sencilla con otros componentes o sistemas informáticos de la organización.
- Puede evolucionar de acuerdo a los requerimientos funcionales y/o tecnológicos que se demanden en el tiempo.
- Se logra reducir el grado de cometer una mala implementación al máximo ya que se orienta hacia las debilidades y fortalezas del equipo de desarrollo.
- Se pueden incluir nuevas tecnologías.
- Se puede incluir y mejorar lo que sea necesario de los proyectos de código abierto a este framework.

3.4.2 Desventajas.

- Se debe asumir el costo de ser el usuario que prueba por primera vez los componentes de la infraestructura.
- Se requiere de al menos un especialista en la arquitectura en que se desarrolla el framework.
- El tiempo de implementación varía de acuerdo a los requerimientos iniciales, y la implementación es un proceso continuo.
- Se limitan las posibilidades de construir un sistema que no esté alineado con la filosofía de la infraestructura, sin realizar un cambio a la misma.

En conclusión esta opción es la más adecuada para aquellas organizaciones en donde no exista un grupo experimentado de desarrolladores pero sí al menos un especialista en la arquitectura, ya que minimizará los errores producidos por la experiencia. Además que puede contar con las ventajas de las otras dos opciones siempre y cuando se esté dispuesto a esperar el tiempo necesario para su construcción, por lo que no se recomienda su uso por primera vez en proyectos de misión crítica donde el tiempo sea un factor importante.

A continuación se presenta en la tabla 3.4.1 donde se muestran las características de las alternativas propuestas y sus respectivas comparaciones.

FrameWork Comercial	Código Abierto	Construcción de FrameWork
El costo de estas herramientas es muy alto, además esta en función del número de servidores y desarrolladores que se incluyan en el proyecto.	El costo de implementación es mínimo.	El usuario asume el costo del desarrollo total, cuando es por primera vez.
La migración se vuelve muy complicada, ya que generalmente se tienen que rehacer los componentes para lograr la migración.	Adaptar los componentes lleva consigo mucho tiempo.	La migración se vuelve muy fácil
No se tiene control de la solución, ésta depende totalmente del proveedor. Por lo que solicitar un cambio puede ser muy costoso, se requiere mucho tiempo de anticipación y sin tener la certeza de que finalmente se pueda realizar.	Se tiene control absoluto de la solución para realizar cualquier cambio.	Se tiene el control completo del funcionamiento de la misma. Por lo que se pueden realizar cambios, aumentos y mejoras en el momento en que lo requieran.
El tiempo de implementación es muy corto, generalmente 10 días.	El tiempo de implementación es tardado.	La implementación es muy rápida y sencilla.
Fácil integración con el servidor y con otros sistemas.	Para el caso de componentes Java (J2EE) pueden correr en cualquier servidor de aplicaciones por lo que facilita su portabilidad y migración.	Se puede integrar fácilmente con otros componentes o sistemas informáticos.
La funcionalidad compleja es relativamente sencilla.		

La posibilidad de crecer en un futuro está limitada a la capacidad de la solución.	Generalmente con el tiempo caen en desuso y/o son absorbidas por las soluciones comerciales. Por lo que no hay crecimiento futuro.	Se puede crecer tanto como se requiera tanto en funcionalidad como en tecnología.
La capacidad de framework en algunos casos puede ser muy grande (sobrada) para sus requerimientos.	La capacidad de framework en algunos casos puede ser muy escasa para los requerimientos.	La capacidad del framework es la requerida, ya que se hace de acuerdo a las necesidades y requerimientos solicitados.
La curva de aprendizaje es grande, ya que se tiene que capacitar a todos y cada uno de los desarrolladores, además que esto incrementa el costo de la solución.	La curva de aprendizaje es muy grande, ya que se tiene que realizar un proceso de investigación.	No existe curva de aprendizaje porque es inherente a la construcción del framework.
La probabilidad de que existan errores es mínima, ya que ha sido probada con anterioridad.	La posibilidad de cometer errores es grande, pues cada desarrollo es único.	Se logra reducir el grado de cometer una mala implementación al máximo ya que se orienta hacia las debilidades y fortalezas del equipo de desarrollo.
Existe mucha información para su uso, inclusive capacitación específica.	La documentación para su uso suele ser magra y no siempre está concentrada en una misma fuente de información. No existe capacitación alguna.	La documentación esta disponible y concentrada en una misma fuente de información.
Incluye soporte y asesoría técnica por parte del proveedor.	No existe soporte ni asesoría técnica formal.	Los que construyen el framework, se encargan del soporte y asesoría técnica.
Aunque limitadas, existen pólizas de garantía para estos productos.	No existe garantía de ningún tipo.	No existe garantía de ningún tipo.

Tabla 3.4.1 Comparación entre el uso de un framework comercial, código abierto y la construcción de un framework

3.5 Conclusiones de las alternativas de solución

Comparando los resultados de las ventajas y desventajas que nos ofrecen las tres alternativas de solución, la opción de “Construcción de Framework” es la más adecuada debido a que se puede tener un mejor control por parte de la organización, ya que permite la especialización de los desarrolladores que participan en los equipos de trabajo hacia tareas específicas debido a que se establecen roles bien definidos como por ejemplo, desarrolladores de componentes de infraestructura de framework o arquitectos y desarrolladores de aplicaciones, además del control que se tiene sobre la flexibilidad para modificar los componentes del framework según las necesidades de la organización.

Debido a que se requiere escalabilidad y flexibilidad de compatibilidad entre diversas plataformas o sistemas operativos así como la capacidad de reutilización de componentes ya desarrollados, es adecuado para el desarrollo del framework usar como herramienta de programación orientada a objetos (POO) J2EE porque cumple con todas esas características, además de que existen diversos componentes libres y comerciales desarrollados para este lenguaje, más que sus competidores como PHP y ASP .NET, el primero por ser totalmente open source y el segundo por tratarse de una plataforma comercial al 100%.

Adicionalmente J2EE cuenta con una variedad de librerías de objetos reutilizables para el desarrollo de aplicaciones WEB, como son los componentes de vista Java Server Pages (JSP) y Servlet que se utilizarán como interfaz hacia el cliente, clases de conexión a datos como las pertenecientes a JDBC , los EJB (Enterprise Java Bean) y los componentes de negocio Java Beans. Por el funcionamiento específico de estas herramientas y por la capacidad de reutilización y escalabilidad, éstas cumplen con las características y beneficios que se requieren en la solución del problema planteado, por lo que se elegirá el diseño de los componentes del framework sobre esta herramienta.

4 SOLUCIÓN PROPUESTA

La solución que se propone y desarrolla en este trabajo toma la opción del diseño de un framework basado en estándares J2EE al cual se le ha denominado Infraestructura de Software para Aplicaciones Web (ISAW) que se describirá a detalle en este capítulo.

Cabe aclarar que el objetivo de este documento es detallar el diseño de ISAW y nunca documentar los estándares de J2EE ni de los Servidores de Aplicaciones, por lo que cuando se haga mención de algún tema referente a estos será únicamente como apoyo a la descripción de ISAW.

4.1 Arquitectura

La arquitectura que se plantea para ISAW cubre la filosofía de la arquitectura general Modelo – Vista – Controlador (MVC), este paradigma plantea la división de componentes especializados, la parte de Modelo contendrá aquellos componentes que implementen las reglas puras de negocio, la Vista proveerá la interfaz por medio de la cuál se estará interactuando con los usuarios y por último el Controlador es el que orquestará las transacciones en un sistema e integrará componentes del Modelo y de la Vista.

En un aspecto más particular ISAW implementará la arquitectura general de MVC planteando cuatro capas lógicas principales:

- Capa de Presentación.
- Capa de Control.
- Capa de Reglas de Negocio.
- Capa de Acceso a Fuentes de Información.

La frontera que divide a las cuatro capas mencionadas anteriormente, a veces es difícil de identificar, sobre todo si tenemos en cuenta que en otras arquitecturas no

es muy clara esta definición, a continuación se definen cada una de las capas, lo que permitirá identificar a que capa pertenece cada uno de los elementos que conformarán a ISAW.

4.1.1 Capa de Presentación.

Provee todos los componentes necesarios para construir la interfaz para el usuario final, Esta capa cubrirá lo siguiente:

- Componentes para presentación de la información y contenido.
- Funciones de validación de información solo a nivel sintáctico, es decir, nada que represente una regla de negocio o consulta a fuentes de información de manera directa.
- Componentes para captura y envío de información.

Los componentes J2EE que se integrarán principalmente son JSP (Java Server Pages), Servlets y JSP Tags. Depende principalmente de la capa de Control que es con la que interactúa enviando y recibiendo información.

4.1.2 Capa de Control.

Funge como el orquestador de las transacciones del sistema, cuenta con un único componente denominado controlador, el cuál toma como base archivos de configuración para definir los componentes a integrar en cada transacción. Esta capa cubrirá los siguientes aspectos:

- Único punto de acceso a la aplicación.
- Soporte a distintos esquemas de seguridad.
- Mecanismos aislados (no dependientes de su origen) para el acceso a la capa de negocio y fuentes de información.
- Elementos para enviar el contenido e información a la capa de presentación.

- Administración dinámica que permita cambiar el funcionamiento de las operaciones o transacciones de una forma ágil sin necesidad de modificar el código de la aplicación.

El componente principal de J2EE que implementará el controlador es el Servlet. Desde esta capa se invocan todos los componentes de las demás capas de forma directa o indirecta.

4.1.3 Capa de Lógica de Negocio.

Modela de forma programática las reglas de negocio de los sistemas informáticos. Es aquí donde reside el verdadero valor del sistema, ya que es una reproducción de los elementos que componen los procesos de las organizaciones.

Desde aquí se invocan a los componentes de la capa de acceso a fuentes de información para consultar o almacenar datos, y a estos darle el tratamiento pertinente para su consulta.

Aunque no implementará ningún elemento particular de J2EE si utilizara los recursos que le provee la capa de Control, su implementación se realiza principalmente en ciertos modelos particulares de clases de java que se verán más adelante.

4.1.4 Capa de Acceso a Fuentes de Información.

Provee una interfaz única de acceso a las fuentes de información, potencialmente puede conectarse a cualquier fuente como bases de datos relacionales, documentos xml, archivos planos, otros sistemas de información, etc.

Esta capa principalmente abstraerá todo el manejo de estructuras de datos de JDBC (Conectividad a Bases de Datos de Java, conjunto de librerías de Java que

proveen el acceso a los manejadores de bases de datos relacionales), y dejará expuesta solamente la parte necesaria para los desarrolladores minimizando la posibilidad de error debido al mal uso de estos componentes.

Uno de sus objetivos es facilitar la exposición de la información como objetos de negocio en vez de estructuras de datos para facilitar el desarrollo de componentes de reglas de negocio.

4.2 Elementos Generales

Se plantea el diseño de ISAW en componentes que puedan ser reutilizables en varias aplicaciones web. Una aplicación Web se define como el conjunto de clases, librerías, contenido web (páginas html y otros archivos de contenido estático), páginas jsp y librerías de etiquetas (tag libraries); estos elementos deberán estar organizados en una estructura de directorios definida por J2EE y empaquetados en un archivo que podrá ser instalado en un Servidor de Aplicaciones, esto aplica para la mayoría de los casos. En la figura 4.2.1 se muestra una referencia a la estructura de archivos y directorios de una aplicación web.

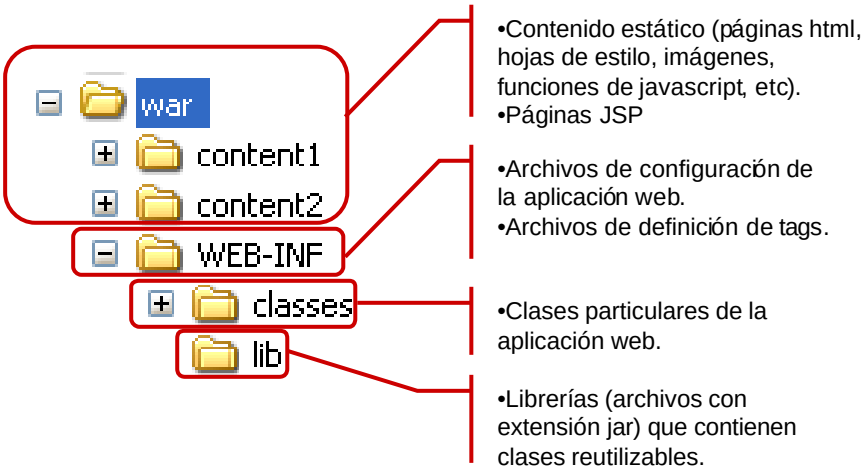


Figura 4.2.1 Estructura de una aplicación web.

ISAW contempla en su diseño distintos elementos para su inclusión dentro de una aplicación web, estos son:

Librería infra.jar, que contendrá todas las clases que modelan los componentes necesarios para cada una de las cuatro capas descritas.
Archivos de definición de tags (archivos con extensión .tld).
Archivos de configuración xml particulares de ISAW (ubicados también en WEB-INF).

Por razones obvias la librería infra.jar es la más compleja ya que contiene los componentes diseñados para ISAW, mientras que la definición de tags y los de configuración son archivos con formato xml necesarios para el funcionamiento de una aplicación web con ISAW.

En la figura 4.2.2 se muestra la ubicación de cada elemento de ISAW en una aplicación web.

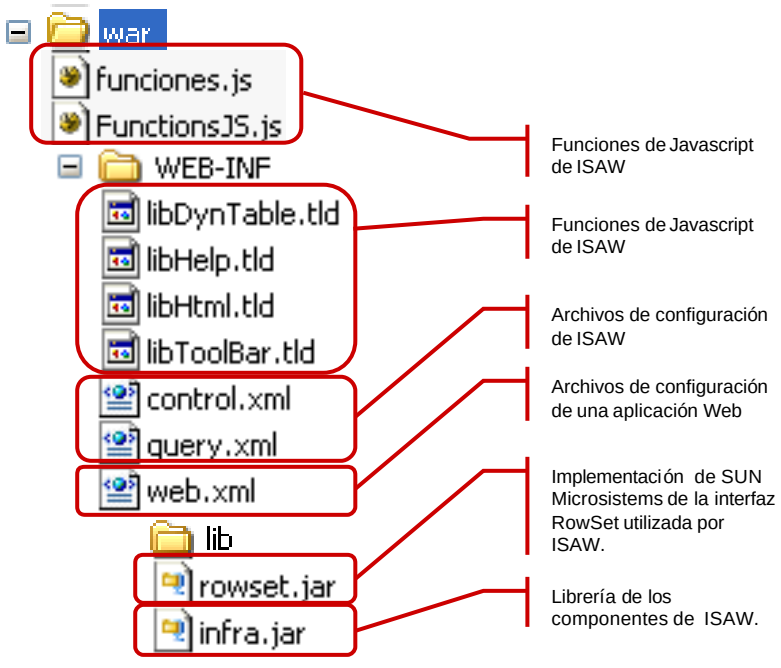


Figura 4.2.2 Ubicación de los elementos de ISAW en una aplicación WEB.

4.3 Modelo Funcional

ISAW plantea un único modelo funcional para una aplicación web, en la figura 4.3.1 se esquematiza cada uno de los componentes de manera funcional, así como la separación lógica de cada una de las capas.

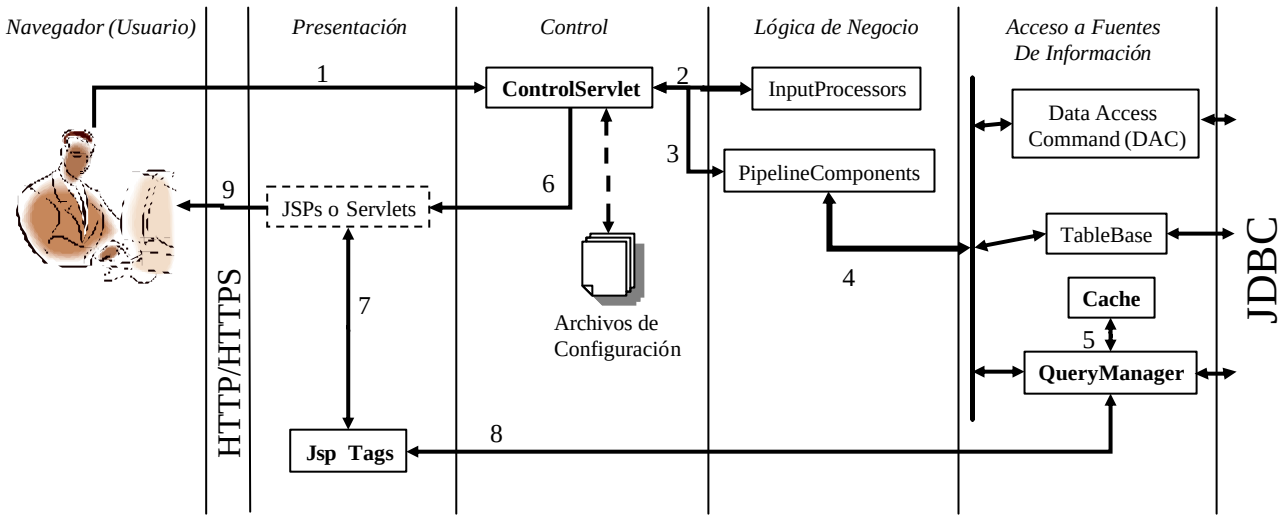


Figura 4.3.1 Esquema funcional de ISAW

Dentro del diagrama se sigue un patrón gráfico para la identificación de los componentes, se enmarcan con una línea punteada aquellos componentes que no formarán parte de ISAW pero que son necesarios para el desarrollo de una aplicación web, los componentes que se escriben en negritas denotan a aquellos componentes que formarán parte de ISAW y serán utilizados tal cual se incluyen en la librería, por último los componentes que se encuentran enmarcados con línea continúa y sin negritas en el texto son aquellos en los que ISAW proveerá una o varias clases de las cuales los desarrolladores tendrán que extender sus propias clases.

Cabe aclarar que estos no son todos los componentes que tendrá ISAW, sin embargo, son los que desde el punto de vista funcional para el desarrollador estarán expuestos, más delante se detallarán todos los componentes que conformarán a ISAW desde un punto de vista más técnico. Por el momento describiremos los componentes principales en esta sección para dar claridad al modelo funcional.

Como se observa en la figura anterior el inicio de cualquier petición o transacción se origina desde un computador que cuente con un navegador; estas peticiones serán siempre atendidas por el controlador denominado ControlServlet ya que se trata de un Servlet, en la mayoría de las implementaciones de arquitecturas MVC se utiliza a un Servlet como controlador debido a la portabilidad y facilidad de manipulación de los recursos de la aplicación Web que proporciona este componente de J2EE.

Desde el ControlServlet se orquestarán todas las acciones subsiguientes, en primer lugar se invocarán los componentes de tipo InputProcessor que tendrán como fin validar los parámetros que se envían por http de acuerdo a las reglas que defina el desarrollador y también inicializará los objetos que serán utilizados durante la transacción.

A continuación se invocarán los componentes de tipo PipelineComponent, cuya funcionalidad será modelar los pasos de los procesos de cada transacción, desde aquí se invocará a la capa de acceso a fuentes de información, ya sea para la lectura o la persistencia de datos. Como lo indica su nombre la filosofía de los mismos es que funjan como tubos de una tubería, donde cada uno representa a un paso de una transacción general, es decir, la tubería. Esto permite una reutilización de estos componentes, por ejemplo, si desarrollamos un PipelineComponent que consulte los datos generales de una persona en una base de datos, este podrá ser reutilizado en la transacción de búsqueda de personas, como también en la impresión de facturas.

Desde los componentes de tipo PipelineComponent se podrá acceder a la capa de accedo a fuentes de información utilizando alguno de los componentes de ésta última.

Los componentes de tipo DataAccessCommand (DAC) serán diseñados para modelar clases que realicen consultas a la capa de datos, la principal ventaja de esta clase será la abtracción completa del lenguaje SQL para los desarrolladores que utilicen la misma, sin embargo, se debe destinar a una persona con experiencia media para su implementación y pruebas.

Otra opción para la consulta de datos es el componente QueryManager el cual funcionará como un ejecutor de sentencias SQL simplificando la ejecución de las mismas a un solo método, tiene como ventaja la posibilidad de paginar los resultados así como el uso de caché, el cuál ayuda a optimizar los recursos de la base de datos. El uso de este componente será muy sencillo ya que no requiere la implementación de alguna clase, bastará con enviar las sentencias SQL, sin embargo, se adquiere la dependencia a las estructuras de información.

El último de los componentes de la capa de acceso a bases de datos es el TableBase el cuál proveerá métodos y propiedades para la implementación de clases que den mantenimiento a las estructuras de información de las bases de datos relacionales. Abstraerá el manejo de los tipos de datos que en ocasiones suelen ser un dolor de cabeza para los desarrolladores.

Una vez concluida la invocación de los componentes de la capa de lógica de negocio el ControlServlet invocará al componente de la capa de presentación que se haya configurado para presentar los resultados de la petición, este componente podrá ser cualquier recurso que pueda ser transmitido por http, y aunque pudiera ser una página html o una imagen esto carecería de sentido ya que este tipo de componentes, llamados de contenido estático no tiene la capacidad de presentar

información proveniente del servidor, por lo que se invocará generalmente a JSPs o Servlets.

Cuando se elige como destino de la transacción a una JSP se podrá hacer uso de los Tags que contendrá ISAW que proveerán una forma sencilla de desplegar información proveniente de las otras capas, así como la integración de parámetros de seguridad que permitan variar el comportamiento de una página bajo distintas condiciones o parámetros.

Es así como al final cada petición que se origine en el navegador del usuario involucrará la ejecución de una transacción definida en los archivos de configuración del ControlServlet y terminará con el envío de html sobre http (o https en caso de las conexiones seguras que manejen SSL), este será enviado por los componentes JSP o Servlets.

4.4 Componentes

ISAW se dividirá en diversos componentes, pero el más completo de todos ellos es el ControlServlet, debido a que es el que fungirá como la parte controladora (Controller) del modelo propuesto.

Cabe aclarar que existirán componentes de uso general que no pueden ser colocados en alguna de las capas, sin embargo, forman parte de ISAW por lo que serán descritos en la sección de componentes comunes.

4.4.1 Capa de Control

Esta capa se encargará de invocar la funcionalidad requerida por el usuario, mediante un mapeo realizado por el componente ControlServlet, que se encargará de llamar a los componentes necesarios para ejecutar dichas acciones que se

traducirán en transacciones que es parte de la capa de negocio (Modelo) y mostrar los resultados al usuario mediante la capa de vista.

4.4.1.1 Configuración

Las peticiones enviadas por el navegador siempre deberán ser capturadas por el servlet de control, el cuál debe ser mapeado a la clase isaw.control.ControlServlet dentro del archivo de configuración de la aplición web (web.xml), por ejemplo:

```
<servlet>
    <servlet-name>ControlServlet</servlet-name>
    <servlet-class> isaw.control.ControlServlet</servlet-class>
    <init-param>
      <param-name>MAIN_DATASOURCE</param-name>
      <param-value>jdbc/DemoPool</param-value>
    </init-param>
    <init-param>
      <param-name>SECURITY_DATASOURCE</param-name>
      <param-value>jdbc/SecPool</param-value>
    </init-param>
    <init-param>
      <param-name>URL_SECURITY_PARAMS</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>SECURITY_VERIFIER_CLASSNAME</param-name>
      <param-value>isaw.control.security.SecDummy</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>ControlServlet</servlet-name>
    <url-pattern>ControlServlet</url-pattern>
</servlet-mapping>
```

Dentro del elemento servlet del archivo web.xml, será necesario incluir los parámetros necesarios para el correcto funcionamiento del ControlServlet, en la tabla 4.4.1 se listan estos parámetros junto con una descripción.

Parámetro	Descripción
MAIN_DATASOURCE	Nombre del Data Source dentro del JNDI del servidor de aplicaciones que se utiliza por omisión para acceder a la base de datos que contiene los datos del negocio, éste se envía a los PipelineComponents y UploadProcessors.

SECURITY_DATASOURCE	Nombre del Data Source dentro del JNDI del servidor que se utiliza para verificar los atributos de seguridad de la aplicación, en caso de no especificarse se utilizará el MAIN_DATASOURCE.
URL_SECURITY_PARAMS	Indica si los atributos de seguridad (v. gr. usuario, perfil, etc.) podrán leerse desde los parámetros del request (true) o se leerán de session (false).
SECURITY_VERIFIER_CLASSNAME	Ruta completa donde se ubica la clase de seguridad que se utilizará para verificar el acceso a los actions y también los niveles de acceso.

Tabla 4.4.1 Parámetros de configuración del ControlServlet

4.4.1.2 Archivo control.xml

Ahora bien el controlador orquestará la forma en que deberán ser ejecutadas las transacciones, que serán llamadas acciones. Dichas acciones se especificarán en el archivo de configuración control.xml que deberá estar localizado en la ruta WEB-INF dentro de cada aplicación Web.

El controlador cargará su configuración cada vez que se instancia el ServletController, la cual obtiene del archivo control.xml. Dentro de este archivo se encuentran todos los elementos necesarios para el funcionamiento de nuestra aplicación. En esta sección describiremos la estructura y contenido de este documento o archivo xml, que será modificado por los desarrolladores de acuerdo a las necesidades cada aplicación.

En primer lugar tenemos la declaración del documento, en donde se definirán ciertos parámetros globales a la aplicación.

```
<?xml version="1.0" encoding="UTF-8"?>
<controller name="demoApp" default="def" reload="false"
initial="initialAction">
```

La primera línea declara el tipo de documento xml, la segunda declara al controlador, sus atributos son el nombre (`name`) el cuál es sólo un distintivo para

diferenciar distintos documentos, el action a ejecutar por omisión cuando no se especifique alguno en particular (`default`). Opcionalmente se puede indicar si se desea que los actions puedan ser invocados más de una vez desde la misma página generada en el navegador, esto por medio del parámetro (reload) que por omisión será (true); este valor será el que se le asignará a cada action en caso de que no se especifique particularmente. El siguiente parámetro opcional es el subaction a ejecutar inicialmente (`initial`) que se ejecutará la primera vez que se invoque una instancia del servlet controlador. En este punto es importante aclarar que el controlador es el encargado de verificar los permisos y seguridad de los action a los que accede el usuario, sin embargo, el acceso a la capa de seguridad deberá ser desarrollado independientemente como se describirá más adelante en la sección de seguridad.

A continuación tenemos la declaración de dos elementos únicos en todo el documento (sólo deben ser declarados una sola ocasión):

```
<norights url="noperm.jsp"/>
<noreload url="noreload.jsp"/>
```

El primero (`norights`) especifica cuál será la página que se desplegará cada vez que el usuario solicite un (`action`) para el cuál no tiene permisos.

El segundo (`noreload`) especificará la página que se desplegará en caso que ese action tenga el atributo (`reload`) asignado a (`false`), y se solicite ese action desde una página que ha sido generada previamente. Esto con el objetivo de evitar que se reenvíen transacciones duplicadas que generalmente ocurre en el uso de la función “Atrás” o “Actualizar” de los navegadores.

A continuación el documento se compone de una lista sin límites de elementos `action`. Este elemento podrá estar repetido tantas veces como se requiera en el documento. A continuación se muestra un ejemplo de la forma más extensa (con más elementos) de un action.

```
<action name="captura" reload="true" mode="alta">
    <parentpage url="Body"/>
    <parenterrorpage url="BodyErr"/>
    <inputprocessor
```

```
class="demo.control.inputprocessors.CargaActionsIP"
verbose="true">
        <parenterrorpage url="BodyErr2"/>
        <propout name="document_catalogs"/>
    </inputprocessor>
    <pipelinecomponent
class="demo.control.pipelines.CalculaAnioIndicePpiPC"
verbose="true">
        <parenterrorpage url="BodyErr2"/>
        <propin name="document_catalogs"/>
        <propout name="document_catalogs"/>
    </pipelinecomponent>
</action>
```

En primer lugar tenemos la definición del action, donde especificamos su nombre (`name`) que es por el cuál la capa de presentación lo solicitará y opcionalmente el atributo (`reload`) que indicará si el action puede ser ejecutado más de una vez desde una página ya generada anteriormente, por omisión toma el valor del atributo especificado en el elemento (`controller`).

El atributo (`mode`) indicará el modo en que será invocado este (`action`), es útil para verificar si el usuario tiene acceso a un (`action`) en particular, en un modo en particular, y también es utilizado por los Tags de la capa de presentación ya que pueden realizar cierta parametrización en base a este valor. Los valores posibles son “alta”, “baja”, “modificación”, “consulta” e “inicial”.

El elemento (`parentpage`) deberá ser declarado solamente una vez dentro de (`action`), indica la página que invocará el controlador una vez concluida la ejecución exitosa del (`action`). Opcionalmente se puede especificar el elemento (`parenterrorpage`) que es equivalente a (`parentpage`),solamente es utilizado en caso de que se genere un error dentro de los elementos del (`action`), en caso de no especificarse se utilizará siempre (`parentpage`).

Siguiendo con el ejemplo existe una serie de elementos `inputprocessor` (IP) y `pipelinecomponent` (PC) que describen las operaciones de negocio a realizar en cada (`action`), estos elementos estarán basados en la implementación de clases que los modelan.

La mínima expresión de un (`action`) será aquella que solamente contiene la declaración del (`action`) con el elemento (`parentpage`), lo cuál es últil para

crear ligas entre páginas sin necesidad de codificarlas en la capa de presentación y sean administradas por el controlador, un ejemplo de esto es el siguiente action:

```
<action name="action1">
  <parentpage url="main_page"/>
</action>
```

4.4.1.3 ControlServlet

Clase `isaw.control.ControlServlet` que extenderá de `javax.servlet.http.HttpServlet` y estará destinada a atender todas las peticiones realizadas a la aplicación web, en su interior realizará varias funciones y dependiendo del momento de su invocación y las condiciones. En la figura 4.4.1.3 se muestra el diseño del ControlServlet

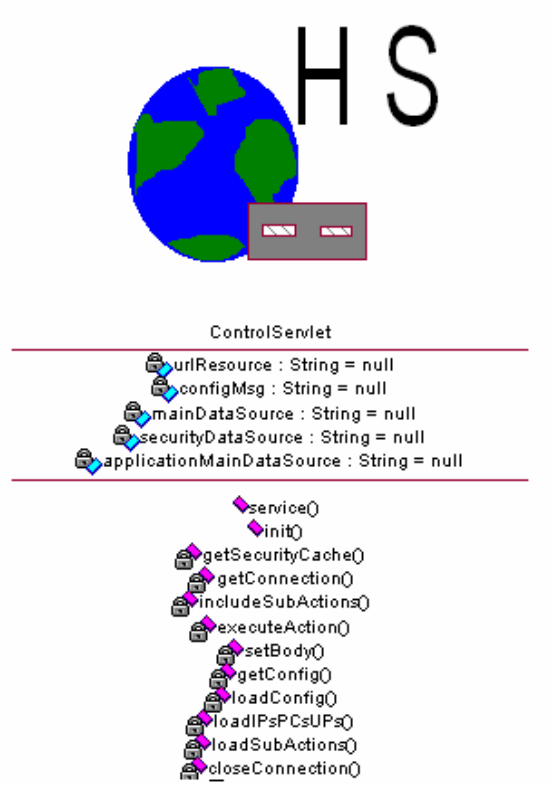


Figura 4.4.1.3 Diseño del ControlServlet

A continuación describiremos los principales métodos que la integrarán.

4.4.1.3.1 Método init

Se invocará únicamente cuando se instancia a este servlet, que generalmente sólo ocurre una vez por aplicación Web (cuando la aplicación Web reside en múltiples JVM no se cumple este comentario). Las operaciones que realiza son:

- Obtiene los DSN que se utilizarán a lo largo de la aplicación.
- Obtiene la clase con que se verificará la seguridad.
- Almacena estos elementos en *application* o *servlet context* para su uso durante la vida de este servlet.

4.4.1.3.2 Método service

Se ejecutará cada vez que se recibe una petición por http a este servlet, y desde este método es de donde se disparará toda la funcionalidad que permitirá a este componente orquestar cada transacción de forma eficiente. Las operaciones que ejecutará son las siguientes:

- Carga la configuración del archivo control.xml (sólo una vez al inicio y cada vez que se forza a recargarla).
- Se verifican y obtienen los atributos de seguridad utilizando la clase de seguridad especificada en el archivo web.xml.
- Ejecución del `subaction` inicial utilizando el método `executeAction`.
- Identifica el `action` a ejecutar.
- Impide que se realice una transacción ya enviada si el atributo `reload` así lo indica.
- Ejecuta el action solicitado utilizando el método `executeAction`.
- Invoca a la url especificada como destino y la envía al navegador, ya sea por una terminación normal o porque se lanzó una excepción.

4.4.1.3.3 Método executeAction

Se invocará únicamente dentro de esta clase y su función será la de mandar llamar a los elementos especificados para completar una transacción. Las operaciones que ejecutará son las siguientes:

- Obtiene todos los elementos a ejecutarse en el `action`.
- Obtiene todos los elementos que se incluyen por medio de `subactions`.
- Ordena todos los elementos de acuerdo a su orden de ejecución.
- Instancia cada una de las clases especificadas en cada elemento y crea un objeto sobre el cuál ejecuta los métodos necesarios para disparar su funcionamiento.
- Administra las transacciones de la base de datos y asegura de esta manera la integridad de la información.
- Ejecuta los `pipelinecomponents` de tipo `finally`.

4.4.1.4 Seguridad

ISAW no planteará la implementación de un esquema de seguridad, debido a que las necesidades de seguridad para cada aplicación que se requieran desarrollar son diferentes en cada organización, por lo cual debe de ser un mecanismo flexible, que permita a los desarrolladores implementar un esquema de seguridad de acuerdo a sus necesidades, por lo que se proveerá de un mecanismo por medio del cual el ControlServlet puede utilizar los atributos de seguridad implementados externamente a ISAW. En la figura 4.4.1.4.1 se muestra el diseño de las clases para el package de seguridad.

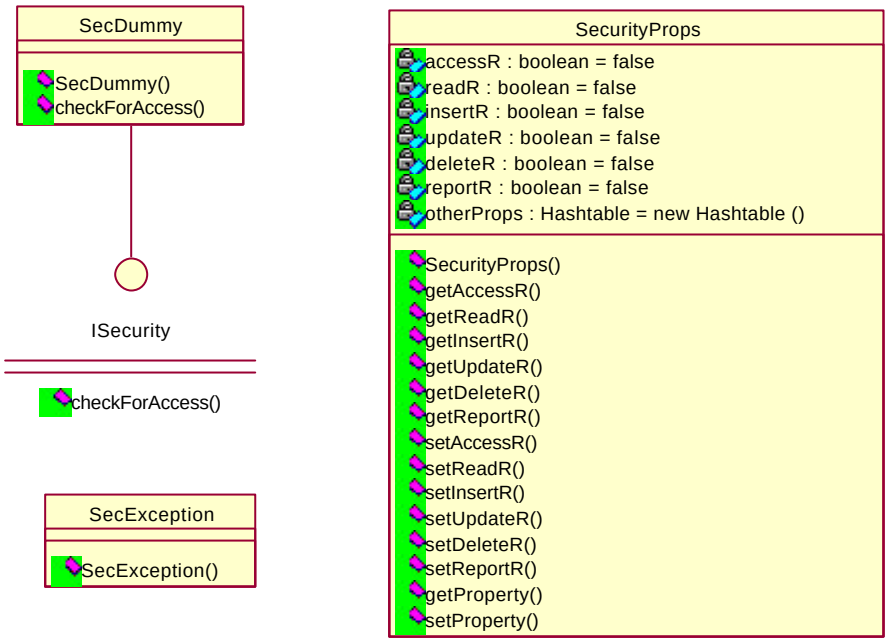


Figura 4.4.1.4.1 Diseño del package de seguridad

Para lograr esto será necesario construir una clase que implemente la interfaz isaw.control.security.ISecurity. Esta interfaz definirá el método checkForAccess, por medio del cual el ControlServlet realizará las consultas necesarias a la capa de seguridad.

Este método deberá regresar un objeto de tipo isaw.control.security.SecProps, y dentro de sus propiedades se detallarán si se tienen permisos de acceso, lectura, inserción, actualización y consulta. Generalmente estos permisos estarán relacionados a los actions, sin embargo, el desarrollador decidirá como implementará el método mencionado para devolver estos valores.

Dentro de ISAW se proporcionará la clase isaw.control.security.SecDummy, la cuál es una implementación de la interfaz de seguridad que siempre devuelve true en todas las propiedades del objeto SecProps. Y en realidad su uso será útil en aplicaciones en las que no se requiere una interfaz a la plataforma de seguridad.

La clase que se utilizará para verificar las propiedades de seguridad deberá ser especificada en la declaración del ControlServlet dentro del archivo web.xml como se indica en la sección Configuración de la Capa de Control.

4.4.2 Capa de Lógica de Negocio

En la funcionalidad de ISAW, existirán dos tipos de componentes que el desarrollador tendrá que implementar extendiendo de las clases InputProcessor y PipelineComponent, sin embargo, existirá un tercer componente llamado UploadProcessor que no es más que un InputProcessor con la funcionalidad necesaria para soportar la recepción de archivos que se envían desde el navegador. A continuación se describirán estos componentes.

4.4.2.1 Input Processor

Los Input Processors formarán parte de la arquitectura de ISAW los cuales serán implementados extendiendo de la clase isaw.control.InputProcessor. En la figura 4.4.2.1.1 se muestra el diseño de los componentes necesarios para la implementación de los input Processors.

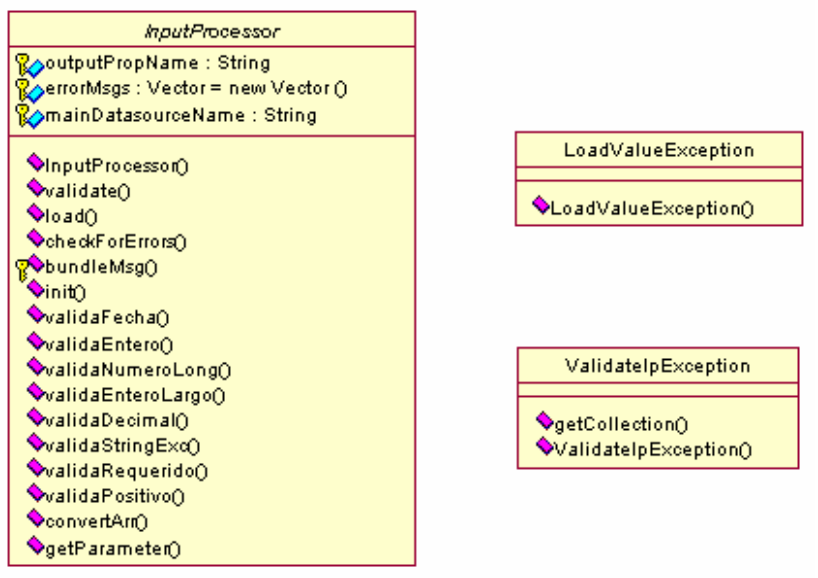


Figura 4.4.2.1.1 Diseño de los inputProcessors

La filosofía del Input Processor será la de leer, validar, trasformar y complementar la información contenida en el request para que pueda ser consumido por un conjunto de Pipeline Components, esto sin necesidad de acceder a la base de datos, por ejemplo, dentro de una página html se debe capturar la fecha en tres campos diferentes:

```
<input type="text" name="fecha_dia" size="2" maxlength="2">
<input type="text" name="fecha_mes" size="2" maxlength="2">
<input type="text" name="fecha_anio" size="4" maxlength="4">
```

La función de un Input Processor particular para esta petición, deberá ser validar la información enviada en estos campos y construir un objeto tipo Date a partir de los mismos, para este caso en particular la clase InputProcessor contiene un método llamado validaFecha, que será el encargado de validar los tres parámetros de tipo String y construir un objeto Date como valor de retorno.

Un valor agregado de la clase InputProcessor es que proveerá un conjunto de métodos para validaciones y conversiones comunes, entre los métodos que estarán disponibles los podemos listar en la tabla 4.4.3 de manera enunciativa:

Método	Descripción
validaFecha()	Validaciones y conversiones de cadenas a fechas.
validaEntero()	Validaciones y conversiones de enteros.
convertArr()	Conversión de arreglos de cadenas a otros tipos de dato.
validaRequerido()	Valida parámetros en el request.

Tabla 4.4.3 Métodos de validación de la clase InputProcessor

La segunda función que desempeñará un Input Processor es la inicialización y carga de un objeto (que la mayoría de las veces es un Bean) al *scope* deseado, el cuál puede ser utilizado por un elemento subsecuente, generalmente un PipeLine Component.

Al desarrollar un Input Processor se deberán implementar dos métodos, el primero es validate() donde realiza la acción de validación y/o conversión. Dentro de éste método el manejo de mensajes de error al usuario se puede realizar por medio de un vector llamado errorMsgs, si es necesario enviar un mensaje de error al usuario indicando alguna condición no satisfactoria en el request o algún otro scope, se puede agregar una cadena a este vector.

Los métodos de validación que proveerá el Input Processor implementan esta funcionalidad internamente. Al finalizar la ejecución de un Input Processor se verificará el vector para decidir si se continúa la ejecución. Este mecanismo no inhabilita la posibilidad de lanzar una excepción, la cuál es igualmente reconocida por el controlador, el tipo de excepción que se debe lanzar es isaw.control.inputprocessors.ValidateIpException. Aunque se recomienda que se utilice el primer mecanismo ya que permite enviar más de un mensaje de error lo cual es útil al momento de atender una petición que involucre más de un valor a ser validado.

El segundo método a implementar será load. Es el encargado de crear y/o poblar un objeto (la mayoría de las veces un Java Bean) con los valores obtenidos del request. Existirán dos formas de recuperar o almacenar el objeto o Bean dentro de este método, la primera será por medio de la propiedad sessionPropName, que contendrá el nombre del atributo donde se almacenará el objeto que se especificará en el archivo de configuración control.xml, cabe aclarar que aunque el nombre de la propiedad comience con la palabra session, se puede utilizar esta propiedad para almacenar el objeto en cualquier *scope* accesible desde un servlet (request, session o application). La segunda forma es únicamente para cuando el objeto en cuestión extiende de ValueObject, en este caso se puede acceder al objeto o almacenarlo por medio de la propiedad valueObject, únicamente se utilizará session para buscar el objeto, y en caso de existir en este alcance se tomará para modificar sus propiedades. En caso de requerir otro *scope* se debe recurrir al primer mecanismo, el cuál es el recomendado.

Es importante aclarar que cuando se ha concluido el manejo del objeto o bean, se debe almacenar en algún *scope* (request, session, o application), por lo que la última línea de éste ejemplo deberá ser implementada en todos los Input Processors que se desarrollen de acuerdo a las necesidades y alcances del atributo.

Las propiedades disponibles para el uso del desarrollador dentro de un Input Processor se muestran en la tabla 4.4.4

Descripción	Propiedad
Request del cliente	HttpServletRequest request
Nombre del atributo dentro de algún scope, donde se almacenará el bean.	String sessionPropName
El bean que se recupera de la sesión. Solamente aplica para aquellos que extienden de ValueObject.	ValueObject valueObject
Sesión.	HttpSession session
Vector de mensajes de error.	Vector errorMsgs
ServletContext heredado del controlador.	ServletContext application
ResourceGetter para el bundle de la aplicación	ResourceGetter mg
Nombre del atributo donde se debe almacenar la salida del IP.	String outputPropName

Tabla 4.4.4 Propiedades de la clase InputProcessor

Para facilitar la obtención de mensajes del bundle de la aplicación se proveerá la función String bundleMsg(String key) que obtiene los mensajes del bundle de la aplicación al proporcionar la llave del mensaje requerido.

4.4.2.2 Pipeline Component

La filosofía de un Pipeline Component, será la de ejecutar las reglas de negocio en forma serializada o en forma de tuberías, de tal forma que su uso sea sencillo. De forma ideal cada Pipeline Component toma un objeto (posiblemente un Bean) como entrada y deja otro objeto como salida (en algunas ocasiones es el mismo objeto de entrada con algunas propiedades modificadas), que puede ser consumido por otro Pipeline y así sucesivamente. La configuración de los Pipeline Components, así como los nombres de los objetos que se almacenan en algún *scope*, es definido en el archivo control.xml.

El único método a implementar en una clase que extiende de isaw.control.pipelines.PipelineComponent es process (figura 4.4.2.2). El nombre en el *scope* dado (request, session o application) del objeto de entrada se puede obtener por medio de la propiedad propIn, y el de salida la propiedad propOut.

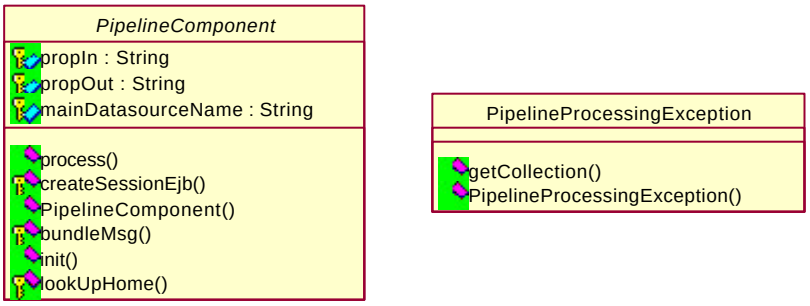


Figura 4.4.2.2 Diseño de los Pipelines

En la tabla 4.4.5 se muestra una lista de las propiedades que se encuentran disponibles en la clase isaw.control.pipelines.PipelineComponent.

Descripción	Propiedad
Nombre del atributo de la sesión, donde se almacenará el bean. El bean que se recupera de la sesión.	String propIn String propOut
Sesión.	HttpSession session

ServletContext heredado del controlador.	ServletContext application
El objeto HttpServletRequest de esta petición.	HttpServletRequest request
Para el acceso al bundle de mensajes de la aplicación.	ResourceGetter mg
Para el acceso al bundle de mensajes de la infraestructura (uso interno).	ResourceGetter infraMg
Para la obtención de la interfaz Home de los EJB.	EJBHomeFactory homeFactory
Conexión a la base de datos a utilizar, es determinada por la configuración del elemento pipelinecomponent en el archivo control.xml	Connection cnn
Nombre del Data Source en el JNDI de la conección a la base de datos principal especificada en el archivo web.xml	String mainDatasourceName

Tabla 4.4.5 Propiedades de la clase PipelineComponent

Al igual que los Input Processors se provee la función String bundleMsg(String key) para el acceso rápido al bundle de mensajes de la aplicación:

Los PCs están también diseñados para acceder a la capa de negocio que en ciertos patrones de la arquitectura J2EE residen en el application server como EJBs. Bajo este tipo de arquitecturas se publican EJBs de sesión llamados façade, los cuales agrupan la funcionalidad de varios EJBs de sesión y/o entidad. Ahora bien, para hacer más eficiente la codificación de las llamadas a los EJBs de façade se propone la construcción de un par de métodos que pretenden eliminar principalmente la obtención de la intefaz Home, y crear la interfaz Remote del EJB, estas funciones son:

```
EJBHome lookUpHome(Class homeClass);
EJBObject createSessionEjb(EJBHome home);
```

Para el método lookUpHome se debe proveer la clase de la interfaz Home del EJB para que pueda ser obtenido del EJBHomeFactory, si ocurre algún error o excepción la excepción PipelineProcessiongException es lanzada. Para el método createSessionEjb solamente es requerida la instancia de la clase que implementa la interfaz Home del EJB, la cual es devuelta por lookUpHome o puede ser obtenida por cualquier otro mecanismo que el desarrollador determine adecuado.
Una vez obtenida la interfaz Remote del EJB se pueden hacer invocaciones a los métodos implementados del EJB de fachada. La implementación de este par de métodos reduce el tiempo de respuesta de las aplicaciones.

Aunque uno de los patrones más aceptados es la implementación de EJBs para la capa de negocio, también se puede prescindir de éstos, en cuyo caso los Pipeline Component pueden fungir como elementos de negocio, para lo cuál se recomienda un esquema de DAO (Data Access Objects) para el acceso a base de datos. El patrón del Pipeline Component será el encargado de proveer el acceso a la base de datos, tanto para consultas a la base de datos (utilizando QueryManager y DAC) como para modificaciones a la misma (por medio de TableBase). La conexión estará disponible en la propiedad cnn, determinada por la configuración del elemento pipelinecomponent en el archivo control.xml.

Se insistirá en que en ningún momento se deberá cerrar el objeto cnn (conexión a la base de datos) ni dar commit o rollback, ya que esto será responsabilidad del Control Servlet. Por lo tanto en ninguna parte de las clases que construya el desarrollador deberá existir una sentencia commit, rollback o close referente a la conexión.

4.4.2.3 UploadProcessor

El objetivo de un UploadProcessor, será la de recibir un *stream* de datos que provienen de un archivo que el browser envía, es decir, provee la funcionalidad necesaria del lado del servidor para "subir" archivos. Una de sus ventajas será abstraer toda la lógica de validación inherente y depositar el contenido del archivo

en propiedades del Upload Processor para que el desarrollador decida el destino de esta información.

Internamente extiende de la clase de un InputProcessor por lo que cuenta con toda la funcionalidad del mismo como son los métodos a implementar (load y validate), sin embargo, implementará otro métodos y mecanismos dentro del mismo, como son la validación del archivo, y acceso a base de datos, los cuales se describen en esta sección. En la figura 4.4.2.3 se muestra el diseño del uploadProcessor y las clases de apoyo para su implementación.

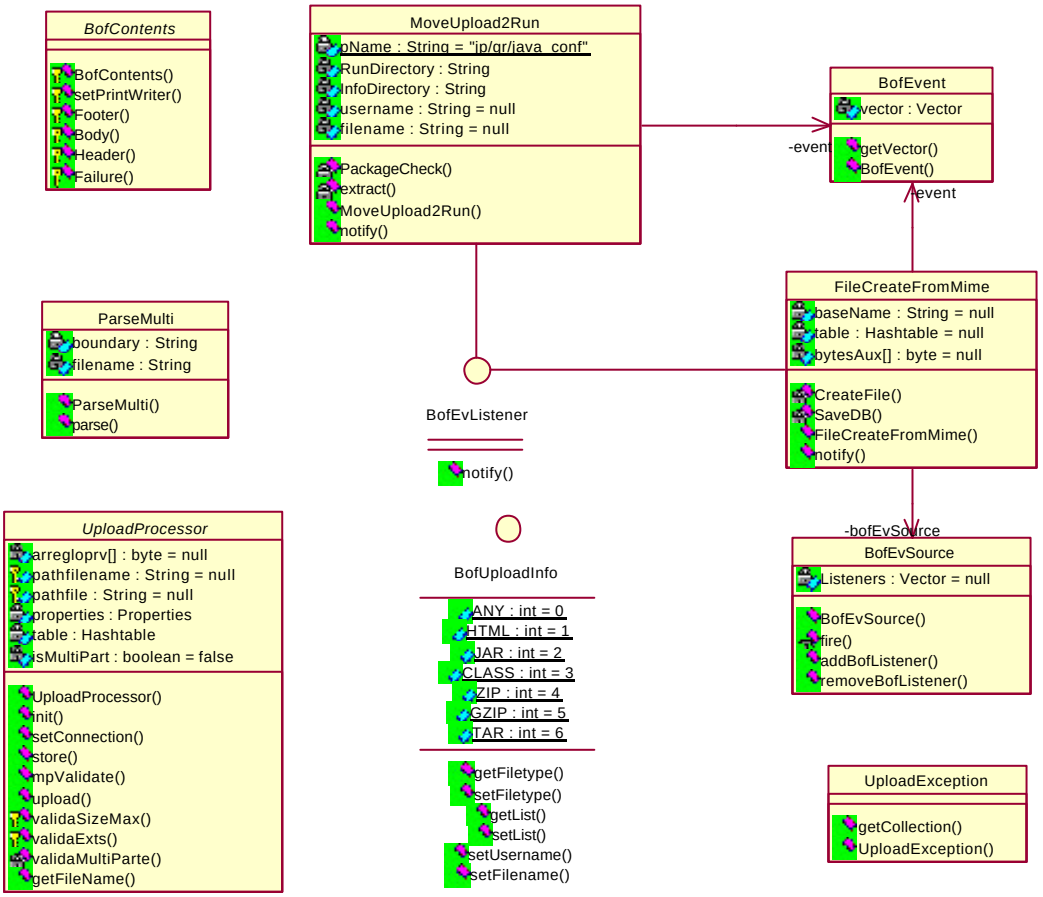


Figura 4.4.2.3 Diseño del UploadProcessor

Cuando se invoque a un Upload Processor, se ejecutará la siguiente secuencia de métodos:

- Valida que el request sea válido. Es decir que el enctype del elemento HTML form que envía el archivo sea "multipart/form-data".
- validate. Este método es implementado por el desarrollador y es descrito en la sección del Input Processor.
- load. Este método es implementado por el desarrollador y es descrito en la sección de Input Processor.
- upload. Es ejecutado internamente y será el encargado de colocar el contenido del archivo en un arreglo de bytes en la propiedad arregloprt.
- store. Es implementado por el desarrollador y tiene la característica que durante su ejecución puede acceder a la propiedad cnn que provee una conexión a la base de datos.

Las propiedades que estarán disponibles para el IP también estarán disponibles para el Upload Processor, sin embargo, se implementarán otras propiedades y métodos particulares, los cuales se listan en la tabla 4.4.6

Arreglo donde se almacena el contenido del archivo (Solamente está disponible durante el método store).	byte[] arregloprt;
Conexión a la base de datos a utilizar, es determinada por la configuración del elemento uploadprocessor en el archivo control.xml. (Solamente está disponible durante el método store)	Connection cnn;
Ruta y nombre del archivo que se está trasmitiendo (Esta ruta es de la computadora de donde se transmite, y el servidor de aplicaciones normalmente no puede acceder a esa ruta, por lo que se debe utilizar con carácter de validación o informativo).	String pathfilename;

Tabla 4.4.6 Propiedades de la clase UploadProcessor

Uno de los métodos que proveerá esta clase es el getFileName que proveerá el nombre del archivo a partir de la ruta completa del mismo.
Se insistirá de que en ningún momento se deberá cerrar el objeto cnn (conexión a la base de datos) ni dar commit o rollback, ya que esto será la responsabilidad del Control Servlet. Por lo tanto en ninguna parte de las clases que construya el desarrollador deberá existir una sentencia commit, rollback o close referente a la conexión.

4.4.3 Capa de Acceso a Fuentes de Datos

Esta capa se encargará de la ejecución de consultas y transacciones hacia la base de datos. Contendrá las clases y componentes necesarios para el acceso a datos. En la figura 4.4.3 se muestran dichos componentes, los cuales se explicarán más adelante.

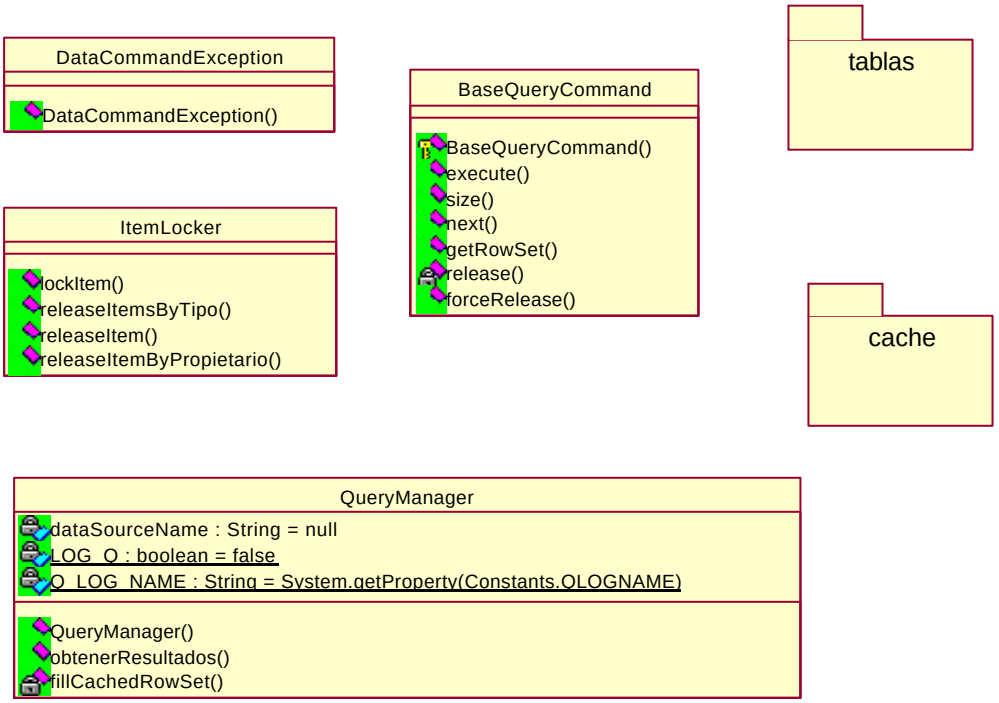


Figura 4.4.3 Diseño del Acceso a Fuentes de Datos

4.4.3.1 Data Access Command (DAC)

Se plantea que la capa transaccional de acceso a las fuentes de información sea por medio de DAO (que será implementado con TableBase dentro de ISAW), sin embargo, dentro de las aplicaciones muchas veces se requiere la consulta de varios registros de información o consultas complejas y variables. Si esto se implementara con DAO resultaría complejo y disminuiría el rendimiento de la aplicación, además DAO se plantea para realizar el manejo de objetos de negocio, mas no consultas masivas.

A partir de estos hechos se pondrá a disposición de los desarrolladores dos estrategias, una de ellas será la del uso de DAC (Data Access Command) y otra será la de utilizar la clase QueryManager (descrito más adelante). Ambas alternativas podrán utilizarse dentro de la aplicación según convenga.

Los DAC son clases de Java que encapsulan la lógica y manejo de excepciones propias de la conexión, preparación y ejecución de consultas a la base de datos, ya que sólo exponen un objeto de negocio con propiedades del negocio de tal forma que si implementáramos un DAC que consultara los saldos de los clientes tendríamos una clase llamada SaldosDAC, con métodos getCliente() y getSaldo(); en caso que requiriéramos pasar parámetros a nuestra consulta sería a través de métodos setters como setPeriodo().

Para implementar un DAC para ejecución de consultas sólo bastará con extender de la clase isaw.data.BaseQueryCommand (la cuál se muestra en la figura 4.4.3.1), agregar un constructor, asignar la consulta e implementar los métodos con los cuales interactuaremos con el resultado. Una de las ventajas al implantar esta solución es que no es necesario cerrar las conexiones o liberar los cursores, la clase por si sola lo hará; aunque esta ventaja también la ofrecerá la clase QueryManager. El uso de DAC permitirá la manipulación de datos en los tipos más

adecuados, esto será muy útil cuando se tratan fechas o números de punto flotante.

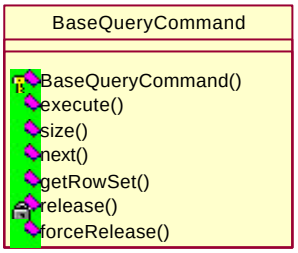


Figura 4.4.3.1 Diseño de la clase BaseQueryCommand

4.4.3.2 QueryManager

La clase isaw.data.QueryManager (figura 4.4.3.2) tendrá como objeto proveer un punto de acceso único al caché implementado dentro de la infraestructura así como dar un fácil manejo a queries de consulta que no requieran parámetros elaborados (en estos casos se recomienda utilizar DACs).

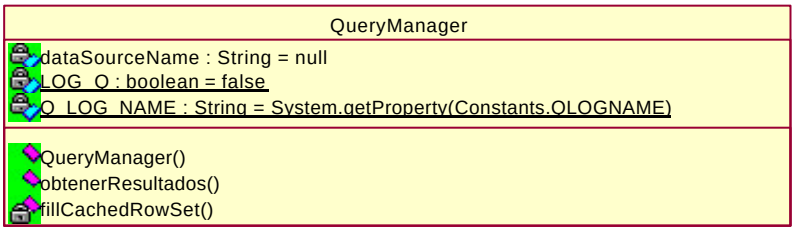


Figura 4.4.3.2 Diseño de la clase QueryManager

Estará diseñado para la ejecución de múltiples consultas y soportar grandes volúmenes de carga (claro que esto está también en función de la capacidad de cómputo). Aunque en principio puede parecer muy atractivo el uso de QueryManager para ejecutar cualquier consulta, se recomendará que sólo se use en aquellas que se requiera alguna característica que ofrece el mismo; ya que por otro lado tendremos el uso de DAC para realizar consultas a las fuentes de información que requieran de un mayor encapsulamiento y manejo de parámetros, aunque ambas estrategias pueden complementarse.

Para obtener una instancia de este objeto será necesario invocar a uno de sus constructores, uno de ellos recibe un java.lang.String que es el Data Source Name en el JNDI del servidor, internamente el QueryManager instanciará una conexión basado en este DSN, la segunda opción es el constructor que recibe un objeto java.sql.Connection que apunta a la base de datos contra la que se desea ejecutar las consultas y es la recomendada para cuando se invoca desde un Pipeline Component o un Upload Processor.

Ofrecerá tres métodos de consulta a la información, los cuales se distinguen uno de otro principalmente por los parámetros que reciben. Analizaremos cada uno de ellos detalladamente.

4.4.3.2.1 Primer Método

```
public CachedRowSetPag obtenerResultados(
                                java.lang.String _strSQL,
                                java.lang.String _strBD,
                                int _desde,
                                int _cuantos,
                                java.lang.String _cache,
                                java.lang.String _tipo)
                                throws java.rmi.RemoteException
```

Los parámetros que recibirá se muestran en la tabla 4.4.7

Parámetro	Descripción
_strSql	Es la consulta a ejecutar contra la base de datos, debe ser un comando SQL bien formado. Tal vez una limitante de este método sea el hecho de concatenar los parámetros como Strings lo cuál a veces no es trivial para tipos fechas, números de alta presición u otros objetos. En caso de que este atributo sea null, el método devolverá null.
_strBD	Es el nombre completo de la referencia a un data source en la configuración del EJB.
_desde _cuantos	Si se especifica un valor mayor a cero en estos dos valores, se regresa una página de los resultados, partiendo del registro indicado por el parámetro _desde y regresando un número de registros igual al

	parámetro _cuantos. En caso de que la consulta no tenga suficientes registros para devolver, se devolverá el máximo número posible de registros.
_cache	Indica si esta consulta se debe recuperar del cache o no, solamente se recuperará del cache en caso de que el valor de este atributo sea "si".
_tipo	En caso de que se utilice el cache este indica que tipo de consulta puede ser si dinámica o de catálogo. Los valores que deben ser especificados son constantes definidas por isaw.Constants.DINAMICO e isaw.Constants.CATALOGO.

Tabla 4.4.7 Parámetros del primer método de acceso al QueryManager

El tipo de objeto que devuelve es isaw.data.cache.CachedRowSetPag, que implementa javax.sql.RowSet, por lo que en los programas clientes que se utilicen este método (al igual que los otros dos) pueden utilizar un objeto tipo javax.sql.RowSet para recibir el resultado lo cuál hace más portable la solución. O en su defecto se puede hacer un cast a java.sql.ResultSet.

En el caso del objeto isaw.data.cache.CachedRowSetPag implementa un atributo llamado tieneMas (que puede ser obtenido por el método getTieneMas()), que en el caso de que se use la paginación, indica si la consulta tiene mas registros que devolver.

En el caso del parámetro _tipo, sirve principalmente para especificar los tiempos de expiración del contenido en el caché, ya que para una consulta dinámica los tiempos de permanencia en el cache serán considerablemente más cortos que para una consulta tipo catálogo. Estos tiempos estarán basados en las constantes CATALOGO_CACHE_TOC,DINAMICO_CACHE_TOC, ATALOGO_CACHE_TOA y DINAMICO_CACHE_TOA de la clase isaw.Constants.

4.4.3.2.2 Segundo Método

```
public CachedRowSetPag obtenerResultados
                        (java.lang.String _strSQL)
                        throws java.rmi.RemoteException
```

Este método se proporcionará como un acceso rápido a la funcionalidad de esta clase. El único parámetro que recibe es la consulta que es del mismo tipo que el

método anterior, este método se ejecuta tomando en cuenta la base de datos especificada en el constructor, sin paginación y sin caché.

4.4.3.2.3 Tercer Método

```
public CachedRowSetPag obtenerResultados
                                (BaseQueryCommand dac,
                                 int _desde,
                                 int _cuantos,
                                 java.lang.String _cache,
                                 java.lang.String _tipo)
```

Este método proveerá la conjunción de las ventajas de las consultas del QueryManager con la estrategia para acceso a la información DAC. Proveerá una estrategia para utilizar la paginación y el caché en las consultas elaboradas que realicen.

La única diferencia en funcionamiento con el primer método es que no se especifica una consulta ni un data source sino un objeto DAC, ya que por si solo contiene la consulta y el datasource contra el que debe ser ejecutado.

Ahora bien la forma de utilizarlo será crear normalmente el DAC, llenar sus parámetros (métodos setters) y dejarlo listo para ser ejecutado. La demás funcionalidad de paginación y caché será la misma. Una de las ventajas que ofrece es que dentro de un DAC podemos manipular con mayor transparencia parámetros diferentes del query.

4.4.3.3 Data Access Object (clase TableBase)

Este patrón se utilizará como alternativa de los EJB de entidad, debido a que a menudo se entra en la controversia de si se deben utilizar los EJB de entidad o no, la cuál no se pretenderá abordar aquí, simplemente se proveerá de una alternativa sencilla y funcional de modificar los objetos en la base de datos abstrayendo la mayoría de las tareas repetitivas que esto conlleva.

Lo que proporcionará ISAW básicamente será una clase llamada isaw.data.dao.TableBase (figura 4.4.3.3) esta abstraerá la mayor parte de la lógica

necesaria par almacenar y recuperar información en la base de datos. Los atributos que contendrá esta clase serán los siguientes:

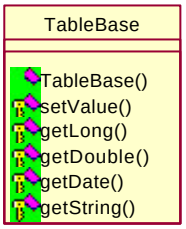


Figura 4.4.3.3 Diseño de la clase TableBase

En la tabla 4.4.8 se muestra la lista de las propiedades que estarán disponibles en la clase isaw.data.dao.TableBase.

Descripción	Propiedad
Conexión a la base de datos a utilizar, es determinada por la configuración del elemento pipelinecomponent en el archivo control.xml	Connection cnn
ResourceGetter que accede al bundle o conjunto de mensajes de la aplicación	ResourceGetter mg

Tabla 4.4.8 Propiedades de la clase TableBase

Uno de los principales problemas a los que nos enfrentamos al momento de almacenar y recuperar información de la base de datos es la compatibilidad de los datos como numéricos con precisión y fecha (entre otros), es por eso que la clase TableBase implementará los métodos para recuperar datos de un ResultSet y para enviarlos a un PreparedStatement que regresan y reciben objetos y no tipos nativos, lo cuál hará más fácil su manejo. Otra de las ventajas de manejar objetos en vez de tipos nativos es que se puede comprobar si es nulo, cosa que no puede ser posible con los datos nativos.

Los métodos que contendrá esta clase se describirán en la tabla 4.4.9

protected java.util.Date getDate(java.sql.ResultSet rs, int index)	Regresa una fecha (Date) del ResultSet, especificando el indice del campo. Regresa null en caso de que el campo esté vacío
protected java.util.Date getDate(java.sql.ResultSet rs, java.lang.String field)	Regresa una fecha (Date) del ResultSet, especificando el nombre del campo. Regresa null en caso de que el campo esté vacío
protected java.lang.Double getDouble(java.sql.ResultSet rs, int index)	Regresa un objeto Double del ResultSet, especificando el indice del campo. Regresa null en caso de que el campo esté vacío
protected java.lang.Double getDouble(java.sql.ResultSet rs, java.lang.String field)	Regresa un objeto Double del ResultSet, especificando el nombre del campo. Regresa null en caso de que el campo esté vacío
protected java.lang.Long getLong(java.sql.ResultSet rs, int index)	Regresa un objeto Long del ResultSet, especificando el indice del campo. Regresa null en caso de que el campo esté vacío
protected java.lang.Long getLong(java.sql.ResultSet rs, java.lang.String field)	Regresa un objeto Long del ResultSet, especificando el nombre del campo. Regresa null en caso de que el campo esté vacío
protected java.lang.String getString(java.sql.ResultSet rs, int index)	Regresa un objeto String del ResultSet, especificando el indice del campo. Regresa null en caso de que el campo esté vacío
protected java.lang.String getString(java.sql.ResultSet rs, java.lang.String field)	Regresa un objeto String del ResultSet, especificando el nombre del campo. Regresa null en caso de que el campo esté vacío
protected void setValue(java.sql.PreparedStatement pst, int index, java.util.Date value)	Sustituye el valor del objeto Date especificado en el indice especificado del objeto PreparedStatement especificado para la ejecución de la transacción
protected void setValue(java.sql.PreparedStatement pst, int index, java.lang.Double value)	Sustituye el valor del objeto Double especificado en el indice especificado del objeto PreparedStatement especificado para la ejecución de la transacción
protected void setValue(java.sql.PreparedStatement pst, int index, java.lang.Long value)	Sustituye el valor del objeto Long especificado en el indice especificado del objeto PreparedStatement especificado para la ejecución de la transacción
protected void setValue(java.sql.PreparedStatement pst, int index, java.lang.String value)	Sustituye el valor del objeto String especificado en el indice especificado del objeto PreparedStatement especificado para la ejecución de la transacción

Tabla 4.4.9 Métodos de la clase TableBase

Se insistirá de que por ningún motivo se deberá cerrar el objeto cnn (conexión a la base de datos) ni dar commit o rollback, debido a que el encargado de esta tarea

será el ControlServlet, por lo tanto en ninguna parte de las clases que construya el desarrollador deberá existir una sentencia commit, rollback o close referente a la conexión.

4.4.4 Capa de Presentación

El elemento más utilizado como elemento de vista será la JSP (Java Server Page), debido a que su implementación es muy similar a la construcción de páginas html. Una de las ventajas de las JSP además de poder incluir código Java que ejecuta el servidor de aplicaciones, es el poder utilizar etiquetas que involucren funcionalidad adicional a las etiquetas clásicas y estáticas de html, que se les denominan tags. Es aquí donde para ISAW se diseñarán componentes agrupados en librerías de tags o etiquetas (en el ambiente técnico se les conoce como taglibs), que se describirán en esta sección.

Además de los tags que se diseñarán para ISAW se podrán integrar otras tags de otros frameworks compatibles con J2EE sin ningún problema, como podrían ser los de Struts del proyecto Apache. En las figuras 4.4.4, 4.4.5 y 4.4.6 se muestra el diseño completo de la capa de presentación.

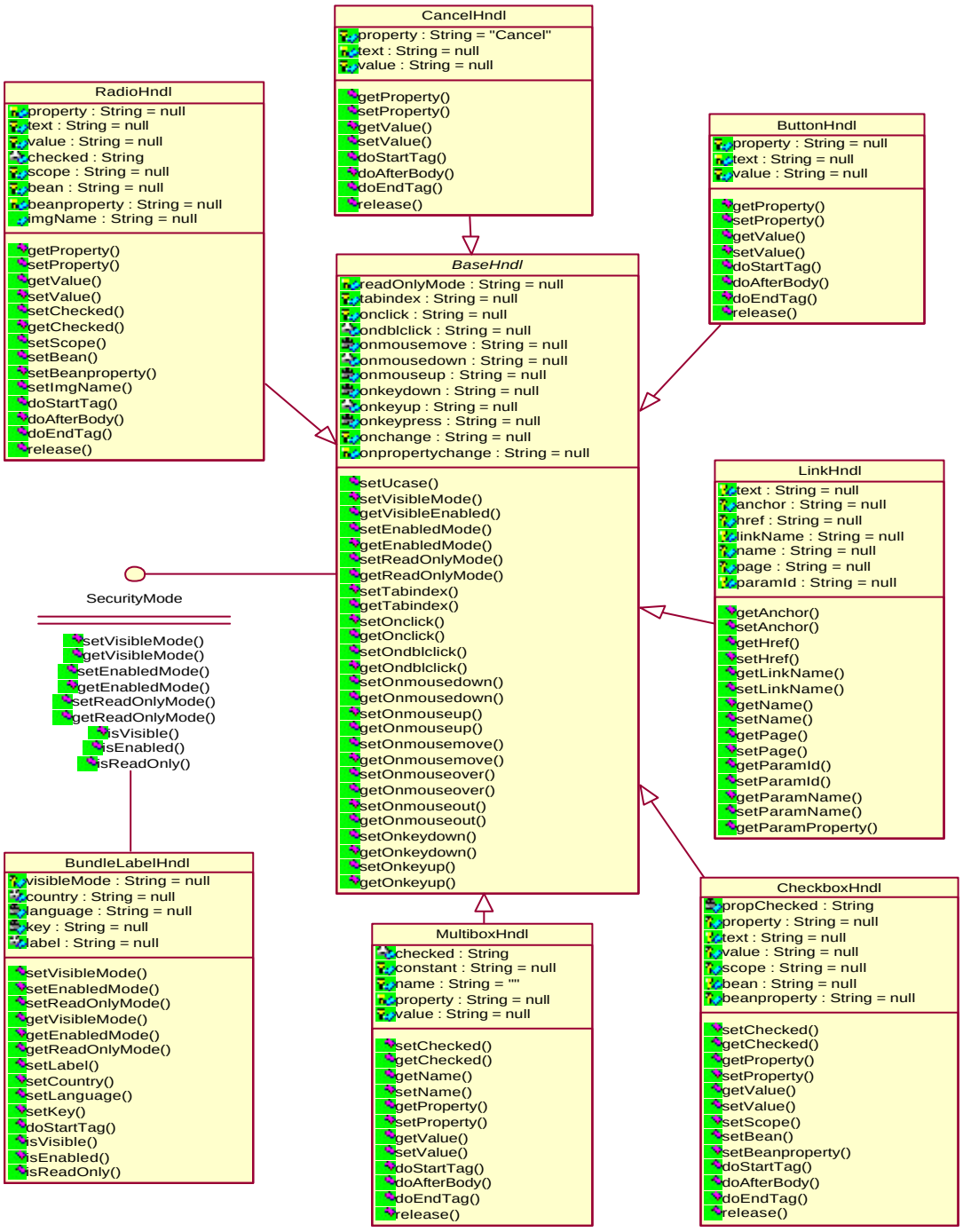


Figura 4.4.4 Diseño de la capa de presentación

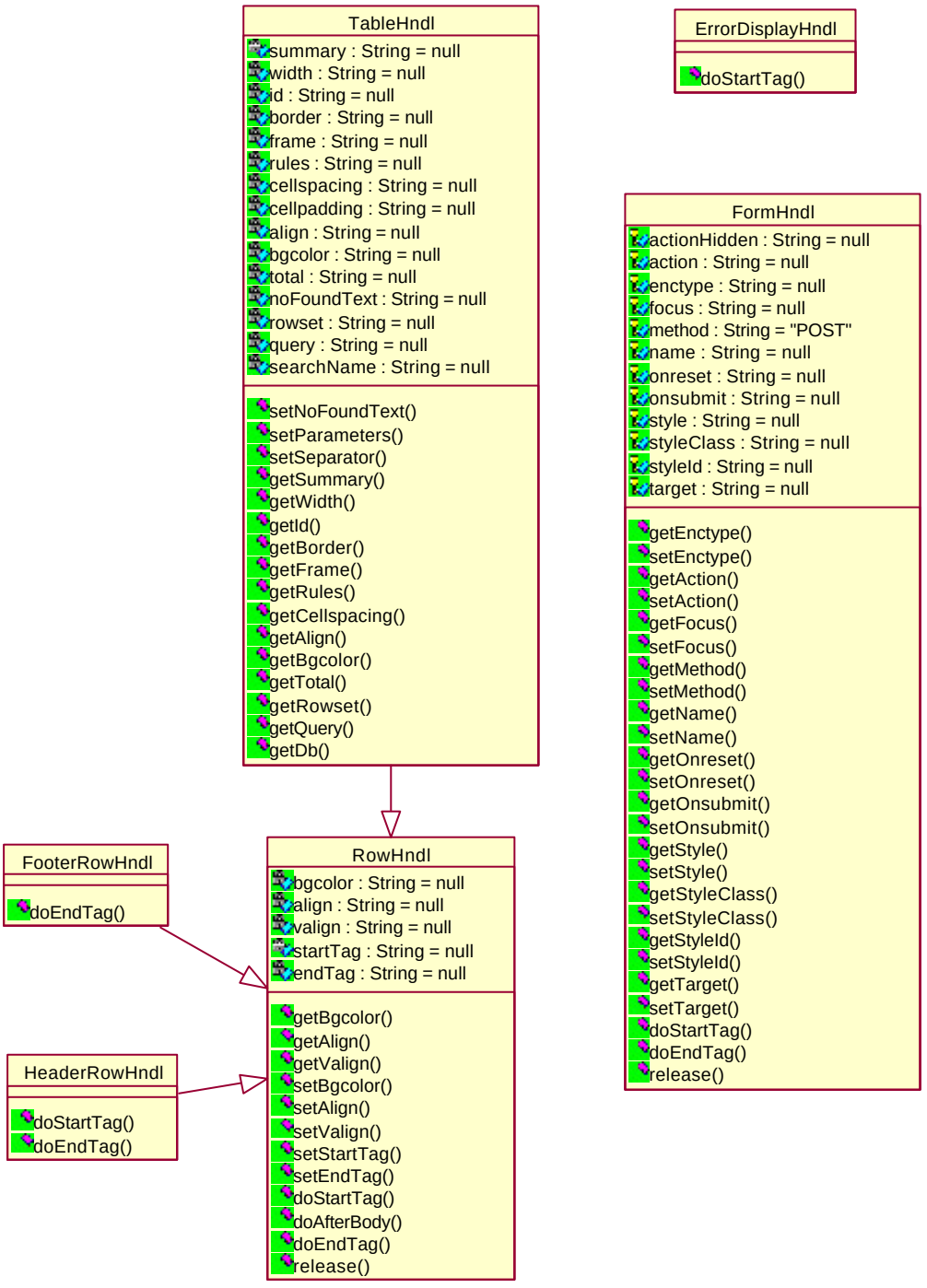


Figura 4.4.5 Diseño de la capa de presentación

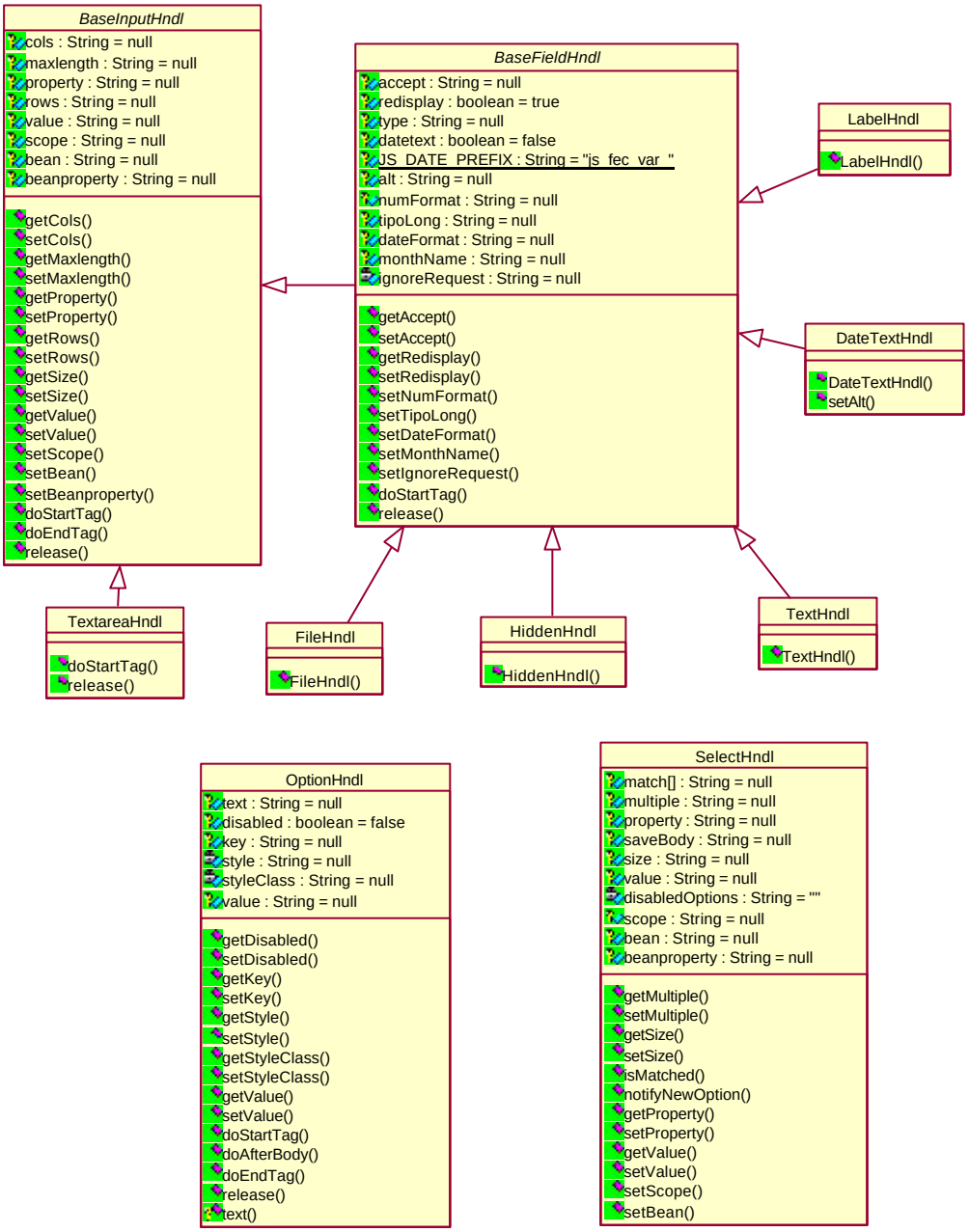


Figura 4.4.6 Diseño de la capa de presentación

4.4.4.1 Librería de Tags para formularios html

Esta librería contendrá las etiquetas que serán utilizadas dentro de los formularios html, tanto para la captura o despliegue de información en una JSP, también incluirá otras etiquetas genéricas que pueden ser de utilidad en la creación de interfaces basadas en HTML.

Muchas de las etiquetas de esta librería arrojarán una excepción de tipo javax.servlet.jsp.JspException en tiempo de ejecución cuando son utilizados incorrectamente (como el uso de una combinación incorrecta de atributos).

Todos los atributos de los tags de esta librería podrán ser generados en tiempo de ejecución (Runtime expression).

En la tabla 4.4.4.1 se describirán más a detalle la funcionalidad de los Tags.

Tag	Descripción
Base	Esta etiqueta o tag permitirá calcular las referencias URL relativas en la página en base URL real de la JSP que contiene a este tag, en vez de la URL desde la cual fue enviada (generalmente es la URL del ControlServlet).
Basescripts	Agregará varias funciones de JavaScript para la invocación de actions desde las JSP.
Bundlelabel	Desplegará una etiqueta o cadena de un ResourceBundle con la finalidad de mostrar mensajes. Podrá ser utilizado en cualquier JSP sin necesidad de estar anidado en otro tag
Button	Generará un elemento input de tipo html button (botón).
Checkbox	Genera un elemento input de tipo checkbox (caja de verificación).
Checklist	Generará una lista de checkboxes basada en un ResultSet o RowSet cargado en algún alcance o scope del request.
Data	Generará un elemento columna de una tabla HTML (la etiqueta html es <TD>). Siempre deberá estar contenido dentro de un Tag row o header dentro de los cuales podrá existir más de una etiqueta data

Ddsection	Esta etiqueta definirá una región en la página HTML donde se pueden incluir o encerrar elementos HTML con la particula ridad de que podrá mostrarse y ocultarse dicha región que se encuentre delimitada por esta etiqueta, que definirá un recuadro para limitar la región o zona. Se ocultará o se visualizará la región al presionar un icono, botón o imagen y además se le podrá dar un título y un estilo ajustable a la región.
Error	Se encargará de desplegar la descripción de un conjunto de errores preparados por la capa de negocio (Input Processors y Pipeline Components). Si no existen mensajes de error presentes no se desplegará algún mensaje.
File	Generará un elemento HTML de formulario tipo file (archivo) con la finalidad de tener una interface para proporcionar la ruta de un archivo local
Footer	Esta etiqueta generará el pie de una tabla HTML que será generada a partir del tag table
Form	Define un formulario html donde se deben incluir los elementos de captura HTML o etiquetas definidas en este framework. Este Tag generará de manera automática algunos elementos ocultos que serán para uso interno de ISAW, por lo que se deberá utilizar este tag para invocar los action.
Header	Esta etiqueta generará una fila de títulos o encabezados que se utilizarán en una tabla HTML creada usando el tag table. Siempre deberá estar contenido dentro un Tag table. Dentro del Tag table sólo podrá existir un elemento header.
Label	Generá una etiqueta de texto que podrá ser accedida desde un bean
Multibox	Esta etiqueta generará campos de entrada de formularios HTML de tipo checkbox. Este será diferente al tag checkbox porque asumirá que los valores contenidos en este tag serán a partir de un arreglo de objetos (de cualquier tipo primitivo soportado o

	String), y el checkbox es inicializado a "checked" si el valor listado para el atributo "value" esta presente en los valores devueltos por el getter de la propiedad.
Option	Generará un elemento de entrada de formularios HTML de tipo option (opción) para el Tag select que es una lista o un combo de selección de valores. Si el valor del parámetro del request con el mismo nombre de este Tag coincide con el valor especificado, esta opción será marcada como seleccionada. Este tag será solamente válido cuando esta contenido dentro de un Tag select.
Table	Generará un elemento HTML de tipo table. Esta etiqueta creará una tabla basada en el contenido de un objeto de tipo javax.sql.RowSet que podrá ser accedido en algún alcance (scope) o generado por medio de un query.
Tablas dinámicas	Este taglib se agregará a ISAW con la finalidad de generar tablas dinámicas, las cuales son utilizadas para agrupar datos obtenidos a partir de consultas simples, de las que se requiere obtener un cálculo acumulado como suma de datos, conteo, promedio, desviación estándar, entre otros cálculos . Todos los atributos de los tags de esta librería deberán ser generados en tiempo de ejecución (Runtime expression).
Barra de Herramientas	Este taglib contendrá aquellas etiquetas que pretenderán eficientar la generación de barras de herramientas con botones basados en imágenes y que a su vez sean sensibles al paso del puntero del ratón.
Button	Generará un botón en la barra de herramientas. Este elemento deberá estar contenido siempre dentro de un tag bar nuevo y no uno que sea una réplica de otro.

Tabla 4.4.4.1 Tabla de tags

4.4.5 Comunes

En esta sección se describirán aquellos componentes que por sus características no podrán clasificarse en alguna de las capas descritas, debido a que podrán ser utilizados por componentes o clases de diferentes capas.

4.4.5.1 Value Objects (Java Beans)

Dentro de ISAW los objetos que servirán como transporte de información se denominarán Value Objects, que son en concreto los llamados Java Beans.
Para que cumplan su objetivo estos objetos deberán cumplir con el estándar de SUN para Java Beans en cuanto a la nomenclatura de propiedades y métodos, y aunque dicho estándar también abarca manejadores de eventos, en el alcance de ISAW solo los usaremos como objetos de transporte (ValueObjects), esto quiere decir que serán instancias que sólo podrán contener atributos encapsulados que podrá ser accesada o modificada mediante métodos públicos como los métodos que serán para modificar información en los atributos denominados como setters y los métodos para recuperar información de los atributos que son llamados como getters.

Los Java Beans que deberán utilizarse como transporte en los Input Processors, Pipeline Components y la capa de presentación deberán cumplir con el estándar mencionado y además implementar la `interfaz java.io.Serializable`. Para ello se diseñará la clase `isaw.ValueObject` para extender los Java Beans que los desarrolladores requieran.

4.4.5.2 Manejo de Excepciones

Las excepciones en Java juegan un papel muy importante en el flujo de operación de las aplicaciones ya que deben contemplar (o más comúnmente dicho: atrapar) cualquier flujo no estándar o normal de la aplicación, por ejemplo, cuando no

existe conexión a la base de datos o cuando los datos enviados son de longitudes o tipos inválidos, etc.
Por medio de ISAW y en especial con los Input Processors se pretenderá eliminar las excepciones que pudieran ocurrir por medio de la validación de la información, sin embargo, es una tarea complicada y que involucra mucho código en contemplar todos los casos posibles. Es por eso que el uso de excepciones será valioso.

Como parte de este framework se implementará un conjunto de clases que extenderán de `java.lang.Exception`, para agrupar las excepciones principalmente por capa. Esta agrupación proveerá la semilla para que se extiendan excepciones más particulares si se llegaran a requerir. Las excepciones que se implementarán son:

- **BusinessException** Se deberá lanzar desde la capa de negocio (EJBs) a la capa de control (preferentemente lo reciben los PCs).
- **ClientException** Esta excepción se podrá utilizar en cualquier clase del lado del web server.
- **PipelineProcessingException** Esta excepción la deberán lanzar los elementos PC al momento en que son procesados.
- **ValidateIpException** Esta excepción la deberán lanzar los elementos IP al momento en que validan la petición.
- **LoadValueException** Esta excepción la deberán lanzar los elementos IP al momento de poblar y cargar el ValueObject.

Para la obtención de los mensajes o textos de las excepciones se usarán los resource bundles, estos archivos que serán descritos a más detalle en la siguiente sección.

4.4.5.3 Administración de Mensajes (Resource Bundle)

Como parte de una de las estrategias de Java para la concentración de mensajes se encuentran los Resource Bundles que pueden ser implementados como clases o como archivos, no obstante su misión principal es proveer una conjunto de mensajes en forma de recursos que estén desacoplados de la aplicación para su fácil administración y manejo de múltiples idiomas (de ser requerido).

Dentro de ISAW, se remitirán estos resource bundles a dos principalmente, uno de ellos no es accesible para el desarrollador y estará localizado dentro del código del framework. El segundo de ellos formará parte de cada aplicación web que el desarrollador genere.

El archivo de recursos para la aplicación web deberá estar localizado en la ruta `isaw.messages.application` bajo el nombre `Labels.properties`.

El formato de este archivo será muy simple, cada renglón esta compuesto por pares de llaves y valores (`llave=valor`).

A continuación se muestra un fragmento de un archivo de recursos.

```
sip.welcome=Bienvenido al Sistema Integral de Personas
sip.contratantes.captura=Captura de Contratantes
buscador.estados=Búsqueda de Estados
buscador.clasif_cte=Búsqueda de Clasificación de Clientes
error.sip.contratante.situacion=Falta capturar la situación
error.sip.contratante.nombre=Falta capturar el nombre
errors.header=<p class="error"> Error(es):<ul class="error">
errors.footer=</ul></p>
errors.prefix=<li>
errors.sufix=</li>
business.ejb.sip.contratanteaux.invalidparam=El usuario ({0}) y la
aplicacion ({1}) no deben ser nulos o vacios.
```

Para facilitar la obtención de los mensajes de este recurso se implementará la clase `isaw.ResourceGetter`, la cual proveerá métodos para devolver de forma eficiente los mensajes para el web server así como el paso de parámetros para construir mensajes con elementos de tipo {*n*}, donde *n* puede tomar valores enteros partiendo del 0 (cero).

4.4.5.4 Paginación de tablas

Dentro de los Tags de la librería para formularios html, se mencionó quedentro del Tag table que se podrá hacer el uso de propiedades de paginación, sin embargo, existirán actividades adicionales a la especificación de estas propiedades a realizar para implementar la paginación en este elemento. A continuación se describirán las características de distintos elementos para implementar la paginación.

4.4.5.4.1 Paginación – Capa de Acceso a Fuentes de Información

Para implementar la paginación necesitaremos realizar dos cosas, una es que nuestro método reciba los parámetros de registro inicial y número de registros que se les enviará a la clase QueryManager. En el caso de no estar utilizando el DAC como estrategia de consulta de información, sino únicamente el QueryManager sólo se deberá observar que en su llamada se incluyan las propiedades de paginación.

4.4.5.4.2 Paginación – Capa de Presentación

En la capa de presentación bastará con agregar el tag table y asignar las propiedades correspondientes.

Conclusiones

Uno de los principales problemas que se nos presentan en la industria de la Tecnología de Información es el obtener mejores resultados en el desarrollo de software en menor tiempo.

Ubicándonos en la colaboración de los desarrolladores, tenemos que, si analizamos los costos que una empresa tiene que contemplar para el proceso de desarrollo de una aplicación específica, tenemos que para la creación de un sistema depende del alcance de la aplicación requerida, es decir, que tan compleja es la funcionalidad requerida, dependiendo del número de desarrolladores que se dispongan para el proyecto, los tiempos de entrega y el presupuesto asignado para dicho proyecto.

En este caso, tendremos que, a mayor complejidad en la funcionalidad del sistema, mayor será el tiempo de desarrollo y mayor será el costo del sistema, contemplando un número determinado de desarrolladores.

Considerando el desarrollo de un sistema con metodología tradicional y con herramientas open source, tenemos que el costo del desarrollo sería muy alto, ya que reduciría la portabilidad y la confiabilidad del desarrollo, pues no se cuenta con toda la información y el soporte de las herramientas open source.

Si consideramos el caso de la adquisición de un software comercial, como por ejemplo un ERP, primeramente el sistema suele no cubrir todos los requerimientos de las empresas, en algunos casos cuentan con demasiados módulos, de los cuales muchos de ellos no son utilizados por algunas empresas, además de tener costos muy elevados, por lo que esta opción no es viable para organizaciones en las que no se cuenta con un gran presupuesto para su adquisición.

Así pues, podemos decir que, con el diseño de un framework utilizando como herramienta de desarrollo J2EE bajo una estructura ISAW, cumplimos satisfactoriamente nuestros objetivos, cubriendo específicamente los siguientes puntos:

- Modularidad

  El diseño de la arquitectura del framework permitirá que el desarrollador que utilice los componentes pueda dividir su funcionalidad en capas o módulos debido a que se maneja el Modelo Vista Controlador, que nos permite dividir una aplicación en funcionalidades de despliegue o de interface hacia el cliente que se compondrán de aquellos componentes de vista como las páginas JSP y tag libraries; en componentes de negocio (Modelo) donde los desarrolladores pondrán la lógica de programación y reglas necesarias que serán la parte primordial de la funcionalidad que implementarán donde usarán los InputProcessors y PipelineComponent; en componentes de acceso a bases de datos para ejecutar las consultas y transacciones necesarias para recuperar y almacenar información de manera optimizada compuesto por las clases QueryManager y TableBase.

  La estructura ISAW permitirá que las aplicaciones evolucionen de acuerdo con los requerimientos funcionales de las empresas, ya que si se requiere agregar funcionalidad o componentes para una aplicación, solo se tienen que modificar aquellos módulos que correspondan a dicha funcionalidad, sin necesidad de modificar el resto de la codificación de la aplicación, inclusive, si cambian las reglas de negocio de la empresa, éstas se actualizarán únicamente en la capa de Lógica de Negocio.

- Escalabilidad y evolución con el tiempo

El framework por su diseño orientado a objetos y por ocupar el patrón de diseño de arquitectura Modelo Vista Controlador podrá evolucionar y será fácilmente agregarle nuevos componentes y funcionalidades que los desarrolladores necesiten en futuras versiones.

- Portabilidad

Por usar como herramienta de desarrollo J2EE, el framework será portable a diversas plataformas tanto comerciales como de software libre, ya que Java funciona bajo varios sistemas operativos como Windows, Unix y Linux, siempre y cuando exista la maquina virtual de Java y servidores de aplicaciones compatibles con J2EE para dichos sistemas operativos. Por lo mismo el framework podrá ser usado tanto en servidores de aplicaciones comerciales (Oracle IAS, Weblogic, Websphere) como de software libre (Apache Tomcat).

- Confiabilidad

El framework es confiable ya que brinda los mecanismos necesarios para implementar la seguridad a medida que una organización lo requiera, ya que brinda una interface de seguridad que deberá ser desarrollada de acuerdo a las necesidades del desarrollador.

- Disponibilidad

Los componentes básicamente usados en una aplicación WEB se encuentran disponibles en el framework y si existe la necesidad de crear nuevos componentes éstos podrán ser desarrollados por los arquitectos de infraestructura que se encargarán de generarlos o bien se podrán utilizar otros componentes compatibles con J2EE ya sean comerciales o de software libre que podrán ser agregados al framework.

- Reutilizabilidad

El diseño orientado a objetos usado en el framework nos permitirá la reutilización de componentes debido a que muchos de los patrones de diseño usados hacen uso de la herencia y de interfaces. A su vez los programadores podrán desarrollar etiquetas o tags dinámicos que encapsulen una funcionalidad de vista que podrán parametrizar.

- Disminución en tiempo y costo de desarrollo

Con el framework propuesto se obtendrá un desarrollo en menor tiempo, debido a que permite la reutilización de componentes y la división de funcionalidades de la aplicación por capas, lo que permitirá enfocar los esfuerzos de los desarrolladores hacia la construcción de funcionalidades por capas o inclusive a un módulo específico, de tal manera, que no se duplicarían las tareas ni el código. También se cubrirían los requerimientos de las empresas de manera específica y eficiente. Además, el mantenimiento, la actualización del sistema y el crecimiento en cuanto a funcionalidad, se vuelve más fácil de controlar. Esto nos trae como consecuencia la disminución de los costos de desarrollo.

GLOSARIO

APPLICATION Uno de los cuatro alcances o scopes que se pueden acceder en una aplicación web, este en particular permanece vigente durante todo el tiempo que la aplicación web está disponible y en ejecución. También se le conoce como ServletContext.

BEAN Componente que tiene la particularidad de ser reutilizable y así evita programar los distintos componentes uno a uno.

BYTECODE Código intermedio utilizado por los implementadores de lenguajes para reducir la dependencia de hardware. Generalmente cada código de operación tiene una longitud de un byte (entre 0 y 255 bits)

CLASE Estructura que contiene propiedades y métodos para modelar objetos.

DAO Data Access Object. Estrategia de acceso y actualización de las fuentes de información que consiste en mapear cada tabla de una base de datos relacional a una clase que tenga métodos de inserción, actualización y eliminación de datos. En ISAW se implementa esta estrategia con el uso de la clase TableBase.

DATA SOURCE Nombre que se le da al recurso que provee conexiones a las fuentes de datos, forma parte del estándar J2EE.

DSN Data Source Name. Representa todo lo relativo a una fuente de datos configurada por el usuario para conectarse a una base de datos, es la información que se tiene que especificar para que el controlador o driver pueda saber con qué fabricante se va a conectar y la cadena de conexión que tiene que enviarle para establecer la conexión con la fuente de datos ODBC accedida por el proveedor en cuestión.

EJB Enterprise Java Beans. Proporcionan un modelo de componentes distribuido estándar para el lado del servidor. Su objetivo es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, etc.) para centrarse en el desarrollo de la lógica de negocio en sí.

FORM Formulario de HTML donde se incluyen los campos a enviar al servidor como una petición.

FORMULARIO Véase form.

HTML HypertText Markup Language. Es un lenguaje de marcación diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de web

HTTP Hypertext Transfer Protocol, Protocolo de Transferencia de Hipertexto. Es el protocolo usado en cada transacción de la Web, es el sistema mediante el cual se envían las peticiones para acceder a una página web, y la respuesta de esa web, remitiendo la información que se verá en pantalla.

HTTPS Hypertext Transfer Protocol over Secure Socket Layer. Es la versión segura del protocolo HTTP. Utiliza un cifrado basado en SSL para crear un canal más apropiado que HTTP para el tráfico de información sensible.

I18N Internaciolalization. Se le conoce así a la facultad que tienen lenguajes de desarrollo y aplicaciones para mostrar los textos y formatos de forma variable dependiendo del país, región y lenguaje donde se encuentre el usuario.

JAVA Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems para la elaboración de aplicaciones exportables a la red y capaces de operar sobre cualquier plataforma a través, normalmente, de visualizadores WWW. El programa en Java se descarga desde el servidor Web y lo interpreta un programa que se ejecuta en el equipo que contiene el explorador de Web.

JAVA BEAN Clase de java que cumple al menos con la característica de exponer métodos setters y getters para interactuar con sus propiedades privadas. Se utiliza generalmente para modelar elementos de la realidad, por ejemplo, la clase Empleado es un bean que tiene los métodos getNombre, getEdada, setSalario, etc.

JDBC Java Database Connectivity Interfaz de aplicaciones que permite la ejecución de operaciones sobre bases de datos desde Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede.

JNDI Java Namming Directory Index. API estándar para acceder y nombrar directorios que incluye una interfaz de provisión de servicio (SPI) la cual sirve de intermediario con otros proveedores de servicios de nombres/directorios.

JSP Java Server Pages, Tecnología Java que permite a los programadores generar contenido dinámico para web en forma de documentos HTML, XML o de otro tipo. Permiten que el código java y algunas acciones predefinidas sean incluidas en el contenido estático del documento web.

JVM Java Virtual Machine. Máquina Virtual de Java que ejecuta el código resultante de la compilación de un programa escrito en java, conocido como bytecode.

J2EE Java 2 Enterprise Edition. Plataforma Java que habilita soluciones para desarrollo, uso efectivo y manejo multicapas en aplicaciones empresariales.

MVC Model View Controler. Arquitectura de desarrollo de aplicaciones web con gran aceptación a nivel mundial. Consiste en separar los datos de una aplicación, la interfaz de usuario y la lógica de negocio.

OBJETO Instancia de una clase.

ODBC Open DataBase Connectivity. Estándar de acceso a bases de datos de Microsoft. Su objetivo es acceder a cualquier dato de cualquier aplicación sin importar que sistema gestor de bases de datos almacena los datos.

POOL DE CONEXIONES Conjunto de conexiones de bases de datos que pueden ser solicitadas más de una a la vez y cuando ya no se usan se devuelven al pool para estar disponibles. Así se ahorra tiempo de conexión y desconexión.

REQUEST Uno de los cuatro alcances o scopes que se pueden acceder en una aplicación web, este en particular permanece vigente desde que se inicia la petición desde el browser hasta que la aplicación envía la respuesta por http al mismo, desde este se pueden accesar a los parámetros que envía el browser.

RMI Java Remote Method Invocation. Es un mecanismo en Java para invocar un método remotamente. Usarlo provee un mecanismo simple en una aplicación distribuida que solamente comunica servidores codificados para Java.

ROWSET Objeto que establece conexión con una base de datos, ejecuta las consultas y administra el conjunto de resultados. Sólo se usa un Rowset para definir parámetros de consulta y no para acceder a los datos y manipularlos.

SCOPE También conocido como alcance. Dentro de una aplicación web se refiere al nivel donde se desea almacenar o leer la información en tiempo de ejecución, existen cuatro: page, request, session y application.

SERVLET Clase de Java que de acuerdo al estándar J2EE su función es despachar las peticiones a una aplicación web utilizando el protocolo http. Extiende de la clase javax.servlet.http.HttpServlet.

SERVLET CONTEXT Contexto del Servidor. Utilizado para referirse al alcance *application*.

SESSION Uno de los cuatro alcances o scopes que se pueden acceder en una aplicación web, este en particular permanece vigente mientras la sesión que mantiene el browser con el servidor se mantenga.

SQL Structured Query Language. Lenguaje utilizado para la manipulación de datos almacenados en una base de datos relacional.

SSL Secured Socket Layer. Técnica para encriptar la comunicación entre dos puntos utilizando el paradigma de llave privada y llave pública.

TAG Por definición quiere decir etiqueta, dentro del contexto html se refiere a las etiquetas que conforman este lenguaje, sin embargo, dentro de las JSP adquieren la dimensión de componentes que pueden ejecutar código del lado del servidor e invocar cualquier clase de java, por lo que cuando en este documento se hace referencia al término Tag se refiere a ésta última definición.

URI Uniform Resource Identifier. Indicador uniforme de recursos. Identifica unívocamente cualquier recurso accesible en una red. Se conforma de dos partes, un identificador del método de acceso (protocolo) al recurso y el nombre del recurso.

URL Uniform Resource Locator. Localizador Uniforme de Recursos. Cadena de caracteres, de acuerdo a un formato estándar, con la cual se asigna una dirección única a cada uno de los recursos de información disponibles en Internet.

XML Extended Markup Language. Metalenguaje extensible de etiquetas que permite definir la gramática de lenguajes específicos. No es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

Bibliografía

Fayad, M. E., Schmidt, D. C., and Johnson, R. E. ***Building Application Frameworks***. Addison-Wesley Pub Co, 1st edition, 1999

Jayson Falkner, Kevin Jones, ***Servlets and JavaServer Pages – The J2EE Technology Web***, Addison-Wesley, 2004

Larman, UML y Patrones. ***Introducción al análisis y diseño orientado a objetos***, G. Prentice Hall. 1999

Mattsson, M., Bosch, J., and Fayad, M.E. ***Framework Integration Problems, Causes, Solutions***. Communication of the ACM. 1999/Vol.42, No.10.

R. Johnson, R. Vlissides, J. Design Patterns. ***Elements of Reusable Object – Oriented Software***. Gamma, e., Helm. Addison-Wesley, 1994 (Edición de 2003 en castellano)

Stephanie Bodoff, Dale Green, ***J2EE Tutorial***, Addison-Wesley Professional, 2004

Sitios Consultados

http://dev.mysql.com/doc/refman/5.0/es/cj-general-j2ee-concepts.html
http://java.sun.com
http://java.sun.com/blueprints/patterns/MVC.html
http://kataix.umag.cl/~mmarin/topinf/perl.html
http://www.cs.indiana.edu/~cbaray/projects/mvc.html
http://webs.teleprogramadores.com/patrones/
http://www.dlsi.ua.es/asignaturas/dpaa/tema1.pdf
http://www.dofactory.com/patterns/patterns.aspx
http://www.efaber.net/formacion/fp/curso_php/index.html