PROYECTO DE GRADO – AÑO 2003

INFORME

Realizado por: Federico Tobler, Sandra Sayanes **Tutora: Ana Erosa**

RESUMEN DEL TRABAJO

En el presente proyecto se investiga la tecnología J2EE, sus mejores prácticas y la utilización de la herramienta IBM WebSphere Studio Application Developer para el desarrollo de aplicaciones empresariales. Se estudian las ventajas de la utilización de frameworks y se adopta el uso de Struts para la capa web de la aplicación.

Como caso práctico, se realiza el análisis de un sistema de gestión con interfaz Web, para la empresa ISA Ltda. Se aplican técnicas de análisis orientado a objetos, basándose principalmente en la utilización de UML y patrones.

Se diseña un caso de estudio del sistema de gestión, utilizando como metodología el diseño basado en componentes, el cual es independiente de la tecnología. Luego, se realiza la correspondencia de cada capa de la arquitectura lógica a las construcciones J2EE y, dado que Struts no se corresponde en forma exacta con dicha arquitectura, se plantean alternativas para contemplar su funcionamiento. A su vez, se proponen distintas opciones para el manejo de la persistencia de los datos.

Por último, se realiza la implementación del caso de estudio seleccionando una de las alternativas planteadas, y tomando decisiones para implementar la comunicación entre las distintas capas y la composición de casos de uso.

Palabras clave: Java 2 Enterprise Edition (J2EE), Patrones de Diseño, Frameworks, Struts, Metodología de Desarrollo Basado en Componentes (CBD), WebSphere.

ÍNDICE GENERAL

CAPÍTULO 1. INTRODUCCIÓN 5

1.1. MOTIVACIÓN 5
1.2. OBJETIVOS 6
1.3. RESULTADOS ESPERADOS 6
1.4. ORGANIZACIÓN DEL DOCUMENTO 6

CAPÍTULO 2. INVESTIGACIÓN DE TECNOLOGÍAS 9

2.1. INTRODUCCIÓN 9
2.2. J2EE 9
2.2.1. Arquitectura 9
2.2.2. Beneficios 10
2.2.3. Construcciones utilizadas en cada capa 10
2.3. PATRONES DE DISEÑO 12
2.4. FRAMEWORKS 13
2.4.1. Necesidad de un framework 13
2.4.2. Beneficios 13
2.5. STRUTS 14
2.5.1. Características 14
2.5.2. Arquitectura Struts 14
2.5.3. Funcionamiento general 15
2.6. RESULTADOS OBTENIDOS 16
2.7. EVALUACIÓN 16

CAPÍTULO 3. ANÁLISIS DEL SISTEMA DE GESTIÓN 18

3.1. INTRODUCCIÓN 18
3.2. METODOLOGÍA DE ANÁLISIS Y DISEÑO ORIENTADOS A OBJETOS 18
3.3. NECESIDADES RELEVADAS 20
3.4. RESULTADOS OBTENIDOS 20
3.5. EVALUACIÓN 20

CAPÍTULO 4. ALTERNATIVAS DE DISEÑO 21

4.1. INTRODUCCIÓN 21
4.2. METODOLOGÍA DE DESARROLLO BASADO EN COMPONENTES 21
4.2.1. Metodología 21
4.2.2. Arquitectura 24
4.3. CORRESPONDENCIA CON LA PLATAFORMA J2EE Y STRUTS 25
4.3.1. Interfaz de Usuario 26
4.3.2. UI de Diálogos de Usuario y Diálogos de Usuario 26
4.3.3. Servicios del Sistema 29
4.3.4. Servicios del Negocio 29
4.4. EVALUACIÓN 33

CAPÍTULO 5. IMPLEMENTACIÓN DE UN CASO DE ESTUDIO 34

5.1. ALCANCE 34
5.1.1. Diagrama de casos de uso 35
5.1.2. Modelo Conceptual 36
5.2. ARQUITECTURA LÓGICA 37
5.2.1. Capa UI Diálogos de Usuario 38
5.2.2. Capa Diálogos de Usuario 40
5.2.3. Capas de Servicios de Sistema y Servicios de Negocio 45
5.3. EVALUACIÓN 46

CAPÍTULO 6. CASO PRÁCTICO 47

6.1. CASO DE USO ELEGIDO 47
6.1.1. Descripción expandida 47
6.1.2. Máquinas de estados 48
6.2. DISEÑO 49
6.2.1. Si se utilizara Struts en su forma estándar... 49

6.2.2. Siguiendo la metodología de desarrollo basado en componentes *50*
6.3. EVALUACIÓN 52
CAPÍTULO 7. CONCLUSIONES Y TRABAJO FUTURO 53
7.1. TRABAJO REALIZADO Y CUMPLIMIENTO DE OBJETIVOS 53
7.1.1. Investigación J2EE *53*
7.1.2. Análisis del Sistema de Gestión *53*
7.1.3. Diseño e Implementación del Sistema de Gestión *53*
7.2. CAMBIOS EN EL LINEAMIENTO INICIAL 54
7.3. APORTES DEL PROYECTO 55
7.4. TRABAJO FUTURO 55
7.5. RECOMENDACIONES PARA FUTUROS DESARROLLOS J2EE EN LA EMPRESA 56
REFERENCIAS BIBLIOGRÁFICAS 57
GLOSARIO 59
LISTA DE ANEXOS 65

Capítulo 1. Introducción

En este capítulo se describen las causas que motivaron la realización de este proyecto y se plantean los objetivos y resultados esperados del mismo.

Por último, se realiza una descripción de la organización de este documento, indicando el contenido de sus capítulos y apéndices.

1.1. Motivación

Desde el año 1994 la empresa ISA Ltda. es socio de negocios de IBM (IBM Business Partner) y a partir de ese momento ha incorporado a sus productos y servicios dicha tecnología.

Hasta el año pasado, el desarrollo de sistemas en ISA se había orientado mayoritariamente a la tecnología Lotus/Domino y a la creación de aplicaciones que faciliten la colaboración entre personas basadas en dicha plataforma. Desde mediados de dicho año se han comenzado a utilizar productos de la familia IBM WebSphere, como ser WebSphere Portal Server, que permite generar portales de empresas con acceso a todas sus aplicaciones.

Uno de los intereses actuales de la empresa es incorporar, en un futuro cercano, un ambiente integrado de desarrollo para construir aplicaciones Java y J2EE, integrando tecnología Java, Web, XML y soporte para servicios Web. Pensando en dicho objetivo se desea trabajar con el producto WebSphere Studio Application Developer (WSAD), por lo que se ha visto como una necesidad importante el adquirir los conocimientos relacionados con la especificación J2EE y su cumplimiento a través de dicha herramienta de desarrollo.

Por otro lado, ISA está instalando filiales en otros países y uno de sus desafíos es lograr la integración de sus sistemas en la Web. Actualmente, la empresa cuenta con varios sistemas propios para manejar su gestión administrativa, los cuales se han ido desarrollando internamente a medida y tienen mucho valor agregado, pero la tarea administrativa se ha vuelto engorrosa debido a que la mayoría de esos sistemas no se encuentran integrados.

La utilización de múltiples sistemas para la gestión de la empresa genera los siguientes inconvenientes:

- El manejo de la gestión se hace cada vez más complicado.
- Existe información duplicada en varios de los sistemas.
- El sistema utilizado para la mayor parte de las necesidades está desarrollado en base al negocio de estaciones de servicio e ISA a tenido que adaptar su funcionamiento a determinadas características de dicho negocio.
- Son sistemas que se han desarrollado hace muchos años y no ofrecen una interfaz amigable.
- No tienen acceso Web.

Sería deseable tener un sistema único de gestión que contemple todas las necesidades de la empresa y que además permita el acceso vía Web, lo cual facilitaría una gestión centralizada de las distintas filiales, evitando de esta forma los costos involucrados en la gestión descentralizada.

Si bien la utilización de sistemas de gestión en plataformas Web puede ser una necesidad de muchas empresas y es posible que existan productos de este tipo, las necesidades particulares de ISA la llevan a desarrollar una solución a medida y aprovechar la oportunidad de utilizar tecnologías de punta.

1.2. Objetivos

El objetivo principal de este proyecto es la investigación de los conceptos involucrados en la especificación J2EE y la conveniencia de su utilización en cada caso. Se desean obtener pautas para el análisis, diseño y desarrollo de aplicaciones empresariales basadas en esta tecnología.

También es clave adquirir conocimientos sobre la utilización de WebSphere para desarrollar aplicaciones que cumplan con los estándares J2EE.

Como segundo objetivo, y para reforzar en forma práctica los conceptos adquiridos, se plantea el análisis, diseño e implementación de un nuevo sistema de gestión para la empresa.

Este nuevo sistema deberá permitir que la gestión administrativa sea operada desde sucursales remotas sin necesidad de implementar una importante infraestructura en cada lugar donde la empresa se instale, por lo cual deberá brindar todas sus funcionalidades a través de un navegador de Internet.

1.3. Resultados esperados

Al final de este proyecto se pretende que los estudiantes hayan adquirido las principales técnicas para la construcción de aplicaciones J2EE y sean capaces de abordar la creación de nuevas aplicaciones basadas en esta plataforma.

Además, se espera obtener una recopilación de pautas y mejores prácticas que permitan la iniciación en el tema a otros integrantes de la empresa.

1.4. Organización del documento

La organización de los próximos capítulos de este documento es la siguiente:

- **Capítulo 2**: Corresponde a la etapa de investigación de tecnologías.

  Describe las ventajas que, luego de nuestro estudio, consideramos proporciona la utilización de patrones de diseño y frameworks en el desarrollo de aplicaciones J2EE.

  Además se realiza una descripción teórica de los conceptos más importantes de la plataforma J2EE y el framework Struts.

  Para el lector no familiarizado con dichas tecnologías, se recomienda la lectura de los anexos correspondientes, en los que se podrá encontrar información más detallada.

- **Capítulo 3**: Se describen las tareas realizadas durante la etapa de análisis del sistema de gestión, la metodología de análisis utilizada y los resultados obtenidos.

  Se plantea la problemática del caso de estudio propuesto inicialmente, su complejidad y la necesidad de acotarlo para este proyecto de grado.

  Además se presenta una metodología de análisis que permite la separación entre casos de uso del negocio y casos de uso del sistema.

- **Capítulo 4**: Se describe la metodología de desarrollo basado en componentes seleccionada para este proyecto y se plantean las dificultades que presenta la utilización de Struts al aplicar dicha metodología.

  El lector que desee obtener más detalles sobre la metodología de desarrollo puede referirse al reporte técnico [VP03].

Este es uno de los capítulos centrales en cuanto a los aportes del proyecto, ya que se realiza un análisis capa por capa de la arquitectura lógica planteada por la metodología y se proponen alternativas de diseño y/o implementación para alinear J2EE y Struts a la misma.

- **Capítulo 5**: Se presentan las decisiones tomadas durante la implementación del caso de estudio.

  Se considera como un capítulo importante, ya que se complementa la arquitectura presentada en el capítulo anterior, profundizando en alguna de sus capas y tomando decisiones para sortear ciertas dificultades, como ser la comunicación entre las capas y la composición de casos de uso.

- **Capítulo 6**: Se describe la aplicación práctica de la metodología de desarrollo junto con las decisiones presentadas en el capítulo anterior. Para ello se toma uno de los casos de uso incluidos en el caso de estudio.

  Este capítulo también es importante porque permite lograr una mayor comprensión de las ideas anteriores.

- **Capítulo 7**: Se plantean las conclusiones de este trabajo, los aportes del mismo y las tareas que se realizarán en el futuro.

  Hacia el final del capítulo podrá encontrarse una sección con las recomendaciones generales para los futuros desarrollos J2EE que se realicen en la empresa.

Conjuntamente con este informe se han entregado documentos anexos con información que lo complementan.

Los anexos están organizados de la siguiente forma:

- Un primer documento, denominado Anexos Teóricos, el cual contiene una serie de anexos que se han ido escribiendo en el transcurso del proyecto, y que pueden utilizarse como apoyo teórico sobre la metodología de análisis y las tecnologías utilizadas.

- Un segundo documento, denominado Anexos Particulares, el cual contiene la documentación que forma parte del resultado de las etapas relativas al análisis, diseño e implementación del sistema, y que brinda un mayor detalle sobre el trabajo realizado.

La descripción del contenido de cada uno de ellos es la siguiente:

Anexos Teóricos:

- **Anexo 1 al 4**: Se describen conceptos y formas de trabajo de acuerdo a la metodología de análisis y diseño planteada por Craig Larman en el libro “UML y Patrones” [Lar99].

- **Anexo 5**: Se realiza un pequeño resumen de la evolución que han tenido las aplicaciones Web desde sus inicios hasta hoy en día, llegando al surgimiento del estándar J2EE.

- **Anexo 6**: Corresponde a un breve análisis de las tecnologías J2EE y .NET realizado al inicio del proyecto. A pesar de que el mismo estaba enfocado a J2EE se resumen las características generales de .NET, ya que podría ser una tecnología alternativa para el desarrollo de aplicaciones Web.

- **Anexo 7**: Se plantean los desafíos que surgen en las aplicaciones empresariales de la actualidad para que las empresas puedan seguir siendo competitivas, y la forma en que J2EE ayuda a lograr ese objetivo.

- **Anexo 8 al 13**: Se detallan los conceptos de J2EE, su arquitectura, cada una de las construcciones que especifica y los patrones de diseño que utiliza.

- **Anexo 14**: Se presenta la idea de frameworks y las ventajas que brindan al desarrollo de aplicaciones.

- **Anexo 15**: Se describen las características del framework Struts, utilizado en este proyecto para la capa web de la aplicación.

Anexos Particulares:

- **Anexo 16**: Corresponde a la especificación de requerimientos del sistema completo de gestión.

- **Anexo 17**: Presenta los casos de uso existentes para todo el sistema de gestión.

- **Anexo 18**: Modelo conceptual de todo el sistema de gestión.

- **Anexo 19**: Se describe en mayor detalle el seguimiento de la metodología basada en componentes al momento de diseñar el caso de estudio.

- **Anexo 20**: Describe detalles de la implementación con WebSphere que, si son tenidos en cuenta en un futuro, permitirán lograr un desarrollo más rápido.

- **Anexo 21**: Contiene las actas de las reuniones mantenidas con el área de gestión de la empresa para realizar el relevamiento de las necesidades del sistema de gestión.

- **Anexo 22**: Para el lector interesado en conocer un poco más acerca de la empresa en la que se realizó el proyecto, en este anexo se reseña brevemente la historia de ISA Ltda.

----- ¤ -----

Capítulo 2. Investigación de Tecnologías

En este capítulo se mencionan las tareas realizadas durante la etapa de investigación de tecnologías. Se realiza una breve descripción de los conceptos involucrados en la plataforma J2EE, las ventajas de la utilización de frameworks y el funcionamiento del framework Struts.

El lector no familiarizado con los conceptos presentados podrá encontrar mayor detalle en los Anexos Teóricos de este informe.

2.1. Introducción

Durante esta etapa se elaboraron documentos con los conceptos más importantes asociados a la tecnología J2EE. Se recabó información de un conjunto de manuales correspondientes a un curso de la versión 4 del WebSphere Studio Application Developer ([WF31102], [WF3502], [WSAD02]) y también se utilizó información disponible en Internet ([Cet02], [Bel03], [Sun00]).

Como gran parte del material utilizado forma parte de un curso de WebSphere, fue posible practicar con esta herramienta a medida que se estudiaba cada concepto.

También se investigaron temas relativos al diseño de aplicaciones J2EE, como por ejemplo patrones de diseño ([Cru03], [Ram02]), mejores prácticas, utilización de frameworks ([SSJ02]) y el funcionamiento del framework Struts ([Cav02]).

2.2. J2EE

Java 2 Enterprise Edition (J2EE) es una especificación abierta diseñada por SUN Microsystems, junto con otro socios de la industria, que permite desarrollar aplicaciones empresariales y servicios Web del lado del servidor. Provee un modelo de programación (API's) que dirige la manera en que deben construirse las aplicaciones.

Distintos proveedores proporcionan herramientas de desarrollo que permiten implementar aplicaciones que cumplan con este estándar.

2.2.1. Arquitectura

La arquitectura J2EE plantea un modelo de n capas: una capa cliente, una capa intermedia (que puede tener una o más subcapas, por ejemplo: subcapa Web que soporta servicios para los clientes y subcapa de negocios que soporta servicios para los componentes de la lógica de negocio) y una capa de fondo (bases de datos y otros sistemas de información).

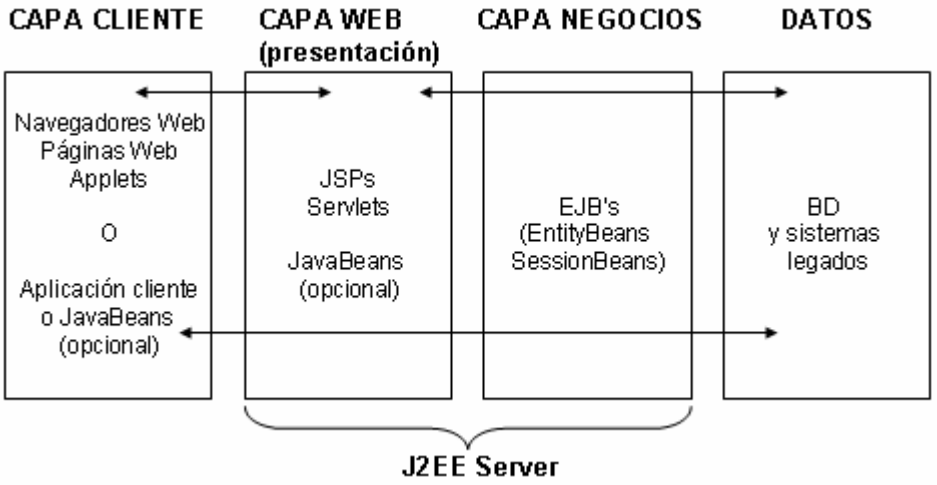


Figura 1 - Arquitectura J2EE

Esta separación en capas se ve facilitada por el uso del patrón de diseño Modelo/Vista/Controlador (MVC), con las siguientes características:

- El modelo
  - Es uno solo
  - Implementa el dominio del problema
  - Maneja toda la lógica del negocio

- La vista
  - Puede haber una o varias
  - Una vista es una "ventana" dentro del modelo
  - Solo provee presentación y manipulación por parte de los usuarios

- El controlador
  - Dependiendo de la aplicación, puede ser uno o varios
  - Determina la solicitud del usuario
  - Valida la existencia de los datos necesarios de acuerdo a la solicitud
  - Invoca los objetos apropiados del negocio

El modelo de desarrollo planteado es basado en componentes, donde el concepto de "contenedor" es fundamental. Los contenedores son ambientes de ejecución estandarizados que proveen determinados servicios a los componentes. Estos servicios estarán disponibles en cualquier plataforma J2EE, de cualquier proveedor.

2.2.2. Beneficios

J2EE permite mejorar la productividad del desarrollo de aplicaciones de varias maneras:

- El uso de componentes permite alcanzar de forma más fácil y flexible las funcionalidades deseadas en la aplicación. También simplifican el mantenimiento de la misma, ya que puede actualizarse o reemplazarse un componente en forma independiente de todo el resto.
- Gracias al alto nivel de estandarización, gran parte del código de la aplicación puede ser generado en forma automática por las herramientas de desarrollo, requiriendo una mínima intervención de parte del programador.
- Los componentes pueden contar con la disponibilidad de servicios estandarizados en el ambiente de ejecución y pueden conectarse dinámicamente con otros componentes.
- El trabajo de los desarrolladores puede dividirse de acuerdo a las capacidades de cada uno de ellos (unos se dedicarán a la presentación, otros a la lógica de negocio, etc.).

2.2.3. Construcciones utilizadas en cada capa

En el caso de una aplicación Web, desde que un cliente realiza una solicitud utilizando un navegador de Internet hasta que recibe su respuesta, ocurre un flujo similar al siguiente entre las distintas capas de la aplicación:

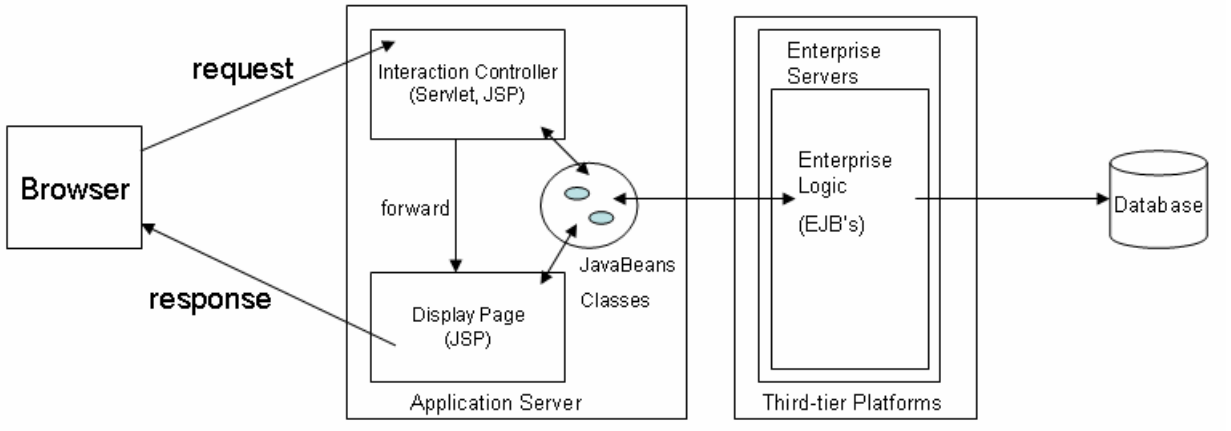


Figura 2 - Flujo Solicitud-Respuesta J2EE

La solicitud del usuario es atendida por la capa controlador, en la cual un "Servlet" es el encargado de localizar y/o crear los objetos de negocio necesarios y redireccionar la solicitud hacia ellos. Dichos objetos de negocio, que se encuentran en la capa modelo, pueden ser "JavaBeans" o "Enterprise JavaBeans (EJBs)". Una vez realizado su trabajo, el servlet obtiene el resultado y envía el control hacia la vista. En ella se utiliza una "JavaServer Page (JSP)" para desplegar contenido dinámico o contenido estático (HTML, DHTML, JavaScript y/o XML).

A continuación se realiza una breve descripción de cada una de las construcciones J2EE mencionadas en el párrafo anterior:

2.2.3.1. Servlets

Un servlet es una clase Java con determinadas particularidades, ya que debe extender *javax.servlet.http.HttpServlet* y por lo tanto hereda sus métodos.

Es un componente Web que corre en la capa del servidor (no del lado del cliente) y que es manejado por el contenedor Web. Se focaliza en el control de la lógica de la aplicación.

Las tareas que realiza el servlet funcionando como controlador son las siguientes:

a. Chequea precondiciones
   - que la solicitud sea válida según el estado de la sesión (dependiendo de la información disponible en el momento)
   - que la autenticación sea correcta
   - etc

b. Realiza su tarea
   - localiza los objetos de negocio apropiados
   - delega lo que corresponda a ellos
   - selecciona el agente de respuesta

c. Establece las tareas de manejo de estado

2.2.3.2. JSPs

Una JSP es un documento XML o HTML al que se le puede agregar código dinámico. Permite separar la lógica (que está en los servlets) de la presentación (que se incluye en las JSPs).

Internamente, una JSP es guardada como un servlet (server-side scripting) cuya generación es llevada a cabo en tiempo de ejecución y es independiente del programador.

Es importante tener en cuenta que las JSP no son una alternativa a los client-side scripting (javascript), ya que éstos son importantes en cuanto a la performance porque eliminan la interacción con el servidor (por ejemplo, para hacer validaciones de datos introducidos por el usuario no es conveniente ir hacia el servidor a chequear su ingreso).

2.2.3.3. JavaBeans

Un JavaBean es una clase Java que debe cumplir con determinadas características, entre las cuales figuran las siguientes:

- Constructor vacío
- Propiedades get para obtener valores
- Propiedades set para establecer valores
- Implementa una interfaz serializable

Esto hace que los JavaBean implementen la *especificación* de un componente de software (objeto empaquetado con una interfaz estandarizada).

Se utilizan principalmente como contenedores de datos serializables (esto es, objetos que pueden ser convertidos a una secuencia de bytes, que puede utilizarse más tarde para obtener el objeto original, incluso a través de una red).

2.2.3.4. EJBs

Un EJB es un componente del lado del servidor que está formado por varios archivos: interfaces, clases Java y un descriptor de despliegue en XML. Se usa para desarrollar componentes distribuidos de la lógica de negocio.

El contenedor de EJBs es un entorno de software que proporciona servicios a los EJBs (middleware) y controla sus ciclos de vida; es quien permite que los EJBs se ejecuten.

Para cumplir con la especificación J2EE el EJB debe implementar y extender ciertas interfaces exponiendo determinados métodos. Independientemente del número de objetos que formen el EJB, el cliente solo puede acceder a él a través de la única interfaz que debe proporcionar.

El EJB tiene una clase principal, denominada "Enterprise Bean", en la cual el programador debe implementar su código (los métodos de negocio).

La especificación J2EE plantea distintos tipos de EJBs, los cuales permiten cubrir distintas necesidades de la aplicación. En la especificación EJB 1.1 se plantean los siguientes:

- SessionBeans
  - Se utilizan para modelar procesos del negocio
  - Una instancia de un SessionBean está asociada a un único cliente a la vez
  - Tiempo de vida corto (dura mientras existe la conexión cliente-servidor)
  - No sobreviven ante fallos (si el servidor cae sus datos se pierden)
  - No son persistentes (no se guardan en bases de datos)
  - Se dividen en dos subtipos:
    1. Stateless: se utilizan cuando no es necesario mantener datos entre las llamadas a distintos métodos (transacciones simples)
    2. Stateful: son utilizados cuando es necesario mantener el estado entre las distintas invocaciones que se realicen en una misma sesión
- EntityBeans
  - Se utilizan para modelar los datos del negocio
  - Pueden ser utilizados por varios clientes a la vez (la concurrencia en manejada por el contenedor)
  - Tiempo de vida extenso
  - Sobreviven a caídas del servidor
  - Son persistentes (se guardan en bases de datos)
  - Se dividen en los siguientes subtipos:
    1. Container Managed Persistence (CMP): el contenedor es el responsable de mantener el estado y de manejar todos los aspectos de acceso al EJB, permitiendo que el programador se concentre en la lógica de negocio.
    2. Bean Managed Persistence (BMP): el manejo de la persistencia está a cargo del programador

2.3. Patrones de diseño

Al diseñar y desarrollar distintas aplicaciones, nos enfrentamos continuamente a los mismos problemas o a problemas muy similares. Esto hace que tengamos que encontrar una nueva solución a un problema similar en cada ocasión. Se podría ahorrar tiempo y esfuerzo si se tuviera un repositorio que capture las características comunes de esos problemas y las soluciones encontradas para resolverlos correctamente.

En términos sencillos, una de esas soluciones a un problema común es un **patrón de diseño**. Un patrón de diseño describe una solución probada para un problema recurrente.

El uso de estos patrones hace transparente el diseño de una aplicación. Los mismos han sido utilizados exitosamente por desarrolladores de distintas áreas, con lo cual los pro y los contra de cada patrón se

conocen de antemano. Además el uso de patrones relacionados con J2EE agrega la ventaja de mostrar soluciones en términos de las tecnologías de la plataforma J2EE.

Nota: *En los Anexos Teóricos de este informe puede encontrarse un documento que describe los patrones de diseño J2EE más importantes.*

2.4. Frameworks

A raíz de las investigaciones realizadas surgió la recomendación de que en la capa Web de la aplicación, se utilice un framework ya existente y probado, antes que diseñar y construir un framework propio [SSJ02].

2.4.1. Necesidad de un framework

Las capas Web de todas las aplicaciones tienen en común un conjunto de requerimientos básicos que no son provistos en forma nativa por la especificación J2EE.

Un framework se sitúa como capa más alta de la plataforma J2EE, proporcionando funcionalidades comunes de las aplicaciones, tales como despacho de solicitudes, invocación a métodos del modelo, selección y armado de vistas, etc.

El framework tiene clases e interfaces estructuradas, las cuales pueden ser extendidas o implementadas por el programador para proporcionar funciones específicas de la aplicación.

La utilización de un framework hace que las tecnologías de la capa Web sean más fáciles de utilizar y permite que el programador se concentre en la lógica del negocio.

2.4.2. Beneficios

Algunos de los beneficios que brinda un framework a una aplicación son los siguientes:

a. Desacopla la presentación y la lógica en componentes separadas: estimula esa separación porque las interfaces están diseñadas de esa forma
b. Separa los roles de los desarrolladores: generalmente proveen diferentes interfaces para diferentes desarrolladores, permitiendo que puedan trabajar en lo suyo independientemente del trabajo de los demás. Los desarrolladores de componentes de presentación podrán focalizarse en la creación de JSPs utilizando tags personalizados, mientras los desarrolladores de la lógica de negocio, por ejemplo, escribirán manejadores de tags y código del modelo.
c. Proveen un punto central de control: muchos frameworks proveen un rico y personalizable conjunto de elementos, como ser plantillas, localización, control de acceso y login.
d. Facilitan el testeo y mantenimiento: como las interfaces son consistentes, las rutinas automáticas de testeo son fáciles de construir y ejecutar.
e. Ahorro de tiempo: el tiempo no utilizado en el desarrollo del código de la estructura queda disponible para el desarrollo de la lógica de negocio.
f. Facilitan la internacionalización: muchos frameworks soportan estrategias flexibles de internacionalización.
g. Pueden soportar validación de entradas: varios frameworks tienen formas consistentes de especificar las validaciones de los datos introducidos por los usuarios.
h. Estimulan el desarrollo y uso de componentes estándares: permitiendo que los desarrolladores y organizaciones puedan acumular y compartir un conjunto de componentes útiles en distintas aplicaciones.
i. Proveen estabilidad: los frameworks son generalmente creados y mantenidos activamente por grandes organizaciones o grupos y son utilizados y testeados en una instalación grande. El código de un framework tiende a ser más estable que el código propio que se pueda llegar a desarrollar.
j. Proveen elementos necesarios: la adopción de un framework permite beneficiarse de la experiencia de un grupo de expertos en el diseño basado en el patrón modelo/vista/controlador. El framework puede incluir elementos útiles para los cuales el desarrollador no tenga la experiencia o el tiempo para construirlos.

k. Soporte de una comunidad: los frameworks atraen comunidades de usuarios que los utilizan y reportan bugs, proveen servicios de consulta, publican manuales, etc
l. Pueden reducir los costos de tiempo y capacitación: los desarrolladores que están familiarizados con el uso de frameworks suelen ser más rápidos y productivos.
m. Pueden ser compatibles con otras herramientas: algunos frameworks pueden integrarse con herramientas de desarrollo rápido, permitiendo mayor productividad.

2.5. Struts

En particular, se decidió investigar la utilización del framework Apache Struts ([Ant01], [Dav01], [Hol02]), para lo cual se consiguió una versión de WebSphere más reciente (versión 5.1) de la que se disponía hasta el momento, la cual brinda soporte para la utilización de dicho framework (la versión 4 no lo soporta).

Struts es un framework desarrollado por Craig McClanahan y donado a la "Apache Software Foundation" en el año 2000. Se ofrece gratuitamente y es de código abierto.

Está basado en el patrón de diseño Modelo-Vista-Controlador y ayuda a construir aplicaciones Web proporcionando sus propios componentes para el Controlador y brindando algunos tags especiales que pueden utilizarse en la Vista. En cuanto al Modelo, este framework no provee componentes para su implementación; permite que el programador decida la tecnología a utilizar (JavaBeans, EJBs, CORBA, JDO, etc) ya que se integra con todas ellas.

2.5.1. Características

Es altamente configurable y tiene una larga y creciente lista de características, dentro de la cual se incluye un servlet "front controller", clases de acción y mapeo, clases utilitarias para XML, poblado automático de JavaBeans, formularios Web con validación y soporte de internacionalización [SSJ02].

Además incluye un conjunto de tags personalizados para acceder al estado del lado del servidor, creación de HTML, creación de lógica de presentación y plantillas.

2.5.2. Arquitectura Struts

Al utilizar Struts, la arquitectura de la aplicación es similar a la siguiente [WSAD03]:

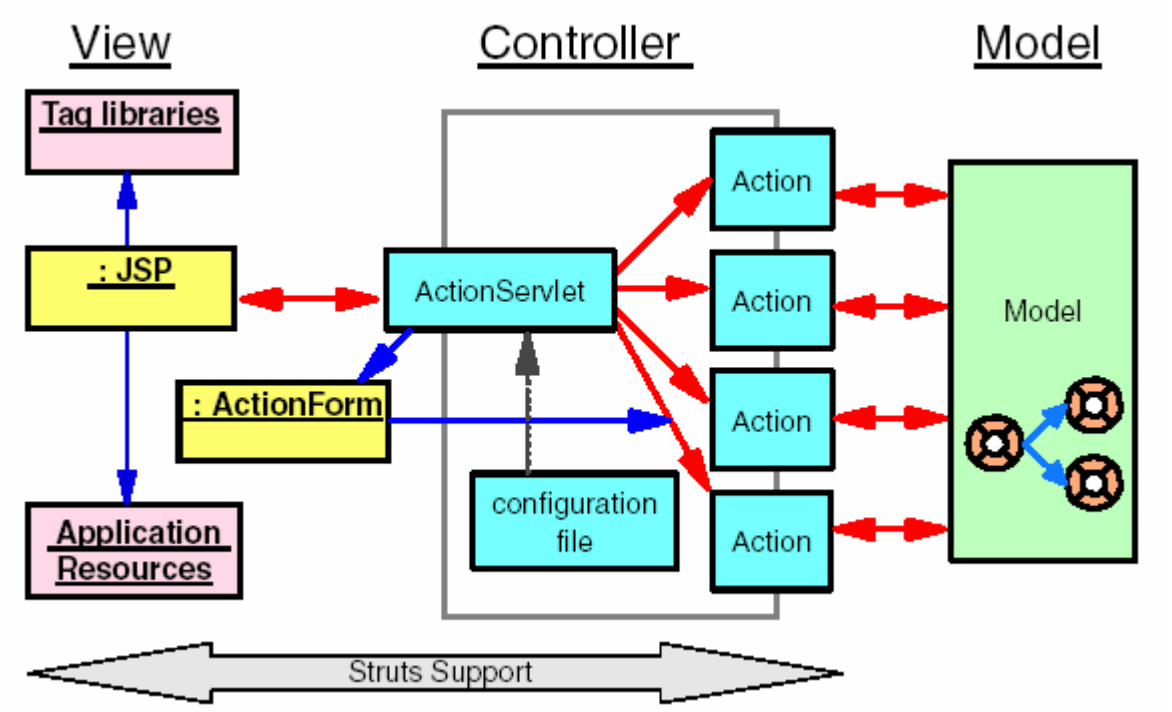


Figura 3 - Arquitectura Struts

La vista tendrá los componentes clásicos (JSP, HTML, JavaScript, Stylesheets, archivos multimedia, etc.) y otros necesarios para el framework:

- **Custom tags:** las páginas JSPs utilizarán tags especiales que se encuentran en las librerías proporcionadas por Struts:
  - HTML: tags para crear el formulario Struts y los campos de entrada.
  - Bean: tags utilizados para acceder a los JavaBeans y a sus propiedades.
  - Logic: tags que permiten realizar recorridas sobre colecciones de objetos y desplegar información de cada uno de ellos.
  - etc.
- **ActionForms:**
  - Es una clase Java que debe implementar el programador extendiendo la clase *org.apache.struts.action.ActionForm* que provee Struts
  - Se utiliza para capturar los datos del usuario desde la solicitud HTTP.
  - Pueden realizarse validaciones sobre los datos introducidos por el usuario.
- **Application Resources:**
  - Struts provee un archivo xml denominado *ApplicationResources.properties*, el cual puede utilizarse para definir constantes y mensajes de error que sean utilizados por las JSPs.

El controlador está formado por los siguientes componentes:

- **ActionServlet:**
  - Es provisto por el framework
  - Se encarga de recibir la solicitud del cliente y determinar qué acción debe ser invocada.
- **Clases Action:**
  - Son clases que debe desarrollar el programador de la aplicación
  - Deben extender la clase *org.apache.struts.action.Action* provista por Struts y sobreescribir el método execute().
  - Actúan como puente entre las solicitudes del usuario y los métodos de negocio.

El archivo xml *struts-config.xml* debe ser utilizado para realizar las configuraciones necesarias para el comportamiento del framework.

2.5.3. Funcionamiento general

Cada solicitud enviada por un cliente (en forma de HTTP Request) contendrá como parámetro un nombre de acción, que Struts utilizará para saber como debe procesar los datos enviados en esa solicitud.

Los nombres de acción se definen en el archivo XML de configuración del framework. Para cada acción se indica además el nombre de la clase Action que se encarga de procesar la acción, y el nombre de la clase ActionForm que encapsula los datos a procesar por la acción.

Como ya se explicó las clases Action y ActionForm son implementadas por el programador.

Las páginas utilizan el tag <html:form> provisto por Struts, cuyo atributo *action* se utiliza para indicar el nombre de la acción asociado a cada página.

Al recibir una solicitud HTTP, Struts busca en su configuración el nombre de la clase Action asociada al nombre de acción enviado por la solicitud, crea dinámicamente un objeto de esa clase e invoca su método *execute()* (puede notarse la aplicación del patrón Command), al cual le pasa el ActionForm que encapsula los datos de la acción.

Esto se ilustra en la siguiente figura [Cav02]:

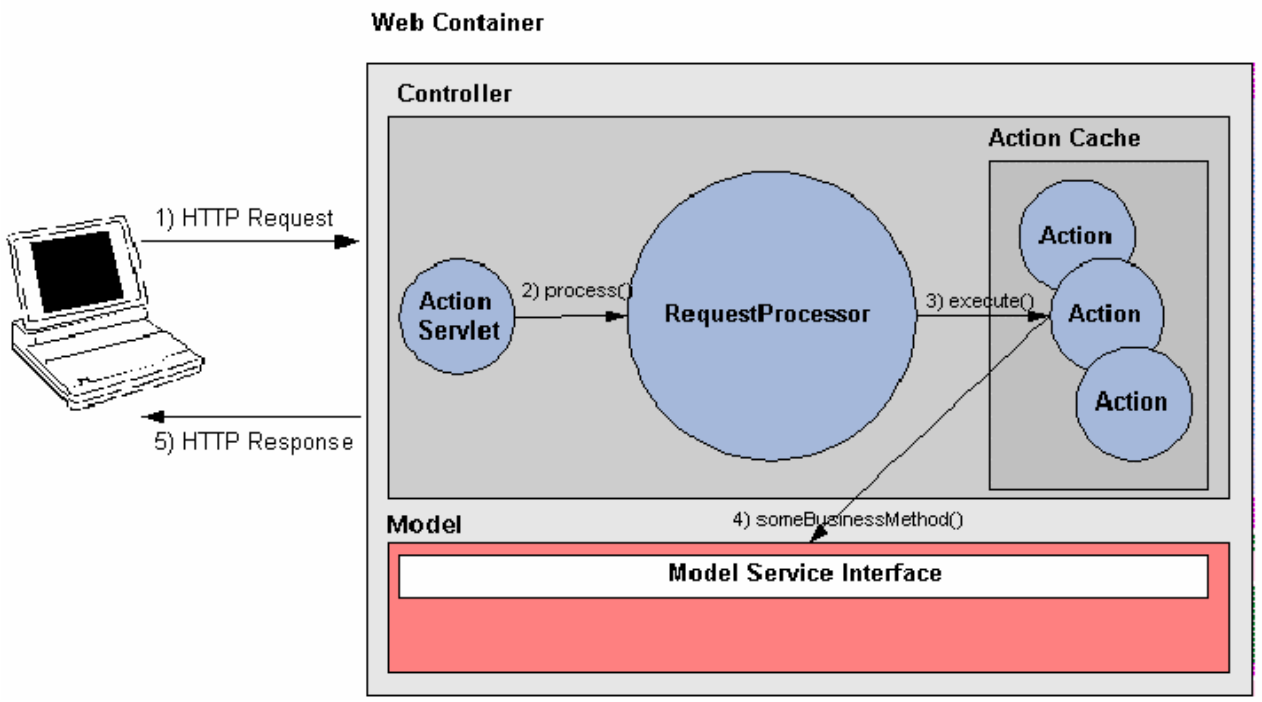


Figura 4 - Funcionamiento básico de Struts

Cuando el método execute de una clase Action finaliza su ejecución, devuelve un ActionForward, que es una referencia a la siguiente página a mostrar. Los ActionForward se definen en el archivo de configuración de Struts. Estos permiten manejar referencias indirectas para las páginas, lo cual mejora la mantenibilidad del código.

2.6. Resultados obtenidos

Como resultado de esta etapa de investigación se adquirieron una gran cantidad de conocimientos, entre los que se destacan:

- Necesidad de un estándar para la construcción de aplicaciones empresariales, que ayude a lograr una clara separación entre las distintas capas de la arquitectura.
- Utilidad de servlets, JSPs, JavaBeans y EJBs.
- Creación de aplicaciones de empresa, y cada uno de los objetos involucrados, en el Application Developer.
- Patrones de diseño J2EE (modelo/vista/controlador, controlador frontal, sesión fachada, objetos de acceso a datos, etc.)
- Ventajas de la utilización de frameworks en la capa Web de las aplicaciones J2EE.
- Funcionamiento de Struts.
- Creación de proyectos Struts en WebSphere.

Se elaboraron una serie de documentos que pueden ser de utilidad a quienes estén interesados en comenzar a trabajar con la tecnología J2EE o con el framework Struts. Dichos documentos se presentan en los Anexos Teóricos del informe.

2.7. Evaluación

La utilización de la tecnología J2EE involucra varias tecnologías subyacentes, como ser: acceso a bases de datos, manejo de transacciones, componentes de empresa distribuidos, y componentes Web, por nombrar algunas. Para comprender las ideas más importantes detrás de J2EE es necesario tener una idea de cada una de esas tecnologías, lo cual hace que sea un poco costosa la iniciación en el desarrollo de este tipo de aplicaciones.

En el mercado actual existen herramientas de desarrollo, entre las cuales está WebSphere Studio Application Developer, que permiten la creación de aplicaciones siguiendo el estándar J2EE y reducen el esfuerzo del programador brindando wizards para la creación de los distintos elementos de diseño.

De todas formas, no alcanza solo con disponer de herramientas de este estilo para lograr una aplicación J2EE de buena calidad. Es importante tener claros los conceptos principales de este estándar antes de comenzar con una implementación.

Además, aunque se tengan claros los conceptos, el programador necesita una guía que lo ayude a decidir cuándo utilizar qué elemento. Para ello existen técnicas y prácticas recomendadas a la hora de desarrollar una aplicación J2EE. Esto ayuda a reducir el tiempo empleado para la implementación, ya que se evita cometer errores comunes, se tienen soluciones a problemas ya abordados por otros desarrolladores, y se pueden aprovechar componentes de soporte existentes.

----- ¤ -----

Capítulo 3. Análisis del Sistema de Gestión

En este capítulo se describen las tareas realizadas durante la etapa de análisis del sistema de gestión y la metodología utilizada.

3.1. Introducción

Para el análisis del sistema de gestión se mantuvieron reuniones con los integrantes de la empresa afectados al proyecto. En esta etapa se analizaron todos los procesos del negocio asociados al nuevo sistema y se relevaron sus requerimientos.

Se decidió analizar el sistema completo por varias razones:

- En principio no se conocía hasta dónde podía llegar su complejidad
- La empresa pretende desarrollar el sistema completo en un futuro próximo (fuera del alcance de este proyecto)
- Conocer todas las necesidades del sistema permite realizar un análisis más "acertado", ya que de lo contrario podrían no tenerse en cuenta aspectos importantes y la continuación del diseño podría impactar fuertemente en el diseño realizado hasta el momento.

Se aplicaron técnicas de análisis orientado a objetos, basándose principalmente en [Lar99]. No se siguió un enfoque único, sino que se usaron ideas de varios trabajos disponibles en Internet, como por ejemplo [Cer03] y [Mol00].

3.2. Metodología de análisis y diseño orientados a objetos

Se comenzó utilizando la metodología de análisis y diseño orientados a objetos propuesta en [Lar99]. La misma está basada en la detección de casos de uso, a partir de los cuales se planifican los ciclos de desarrollo.

Se define un caso de uso como una descripción narrativa de un proceso de dominio, por ejemplo *Obtener libros prestados en una biblioteca*, que describe la secuencia de eventos de un actor (agente externo) que utiliza un sistema para completar un proceso. Los casos de uso no son exactamente los requerimientos ni las especificaciones funcionales, sino que ejemplifican e incluyen tácticamente los requerimientos en las historias que los narran.

En [Lar99] se definen dos tipos de casos de uso, los de alto nivel, que son narraciones sencillas sin mucho nivel de detalle, y los casos expandidos, que son descripciones bastante detalladas de los procesos. También se proporcionan plantillas bastante útiles para la notación de los casos de uso.

Esta metodología no examina todos los aspectos de un proceso de desarrollo completo, sino que se centra en aquellos relacionados con el aprendizaje de habilidades de análisis y diseño orientados a objetos.

Las etapas de macro nivel propuestas para la presentación de una aplicación son:

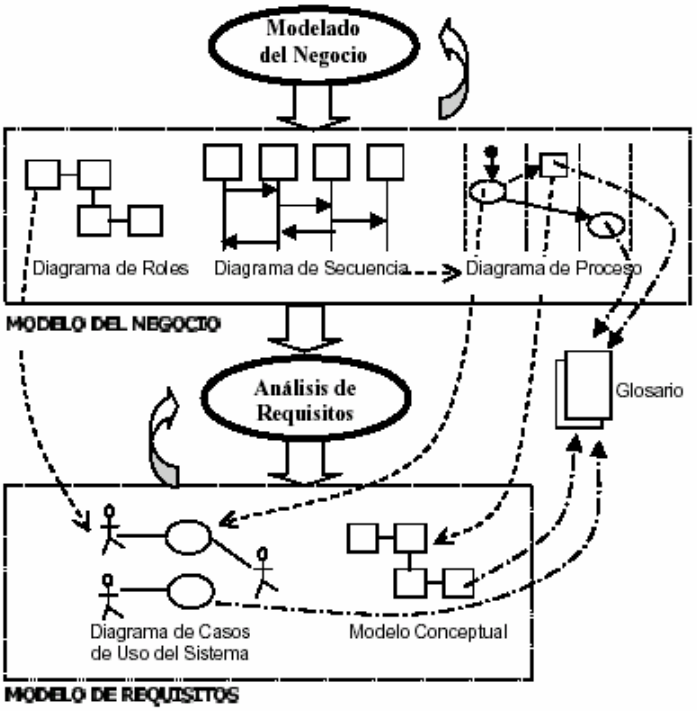
1. **Planeación y elaboración**: Esta fase incluye la concepción inicial, investigación de alternativas, planeación del proyecto, especificación de requerimientos, detección de casos de uso, y creación del modelo conceptual preliminar.
2. **Construcción**: En esta etapa se definen y ejecutan ciclos de desarrollo iterativos, que se organizan a partir de los requerimientos de los casos de uso. Esto es, se asigna un ciclo para implementar uno o más casos de uso o bien sus versiones simplificadas. Los casos de uso deberían clasificarse, y aquellos que ocupen los niveles más altos deberían abordarse en los ciclos iniciales.
3. **Aplicación**: Se realiza la transición de la implementación del sistema a su puesta en producción.

Al aplicar esta metodología para la identificación de los casos de uso, no resultaba sencillo detectar y expresar los casos de uso del sistema bajo estudio. Se vio la necesidad de establecer una separación entre los procesos del negocio y los casos de uso del sistema, la cual no se visualizaba tan claramente en esta propuesta.

Investigando sobre el tema se encontró un trabajo ([Mol00]) en el cual se mostraba de manera bastante explícita dicha separación, y en cierto modo se contemplaban las dificultades encontradas al identificar los casos de uso:

> "*Aunque el éxito de los casos de uso se suele justificar con el hecho de que constituyen una técnica simple e intuitiva, algunos autores señalan las dificultades que entraña la obtención y la especificación de casos de uso verdaderamente útiles, y la falta de consenso sobre cómo organizarlos y manejarlos. Estas son las razones que nos llevan a pensar que es necesario establecer un conjunto de guías para la identificación, descripción y organización de los casos de uso.*"

Dicho trabajo se basa en la *Arquitectura de Tres Modelos de OOram.* En él se propone realizar el modelado del negocio mediante diagramas de actividades UML. Una vez determinados los procesos de negocio de la organización, y descriptos sus flujos de trabajo mediante diagramas de actividades, los casos de uso se licitan y estructuran a partir de las actividades de cada proceso, mientras que las entidades del modelo conceptual se obtienen de los datos que fluyen entre tales actividades. Además, se identifican las reglas del negocio y se incluyen en un glosario como parte de la especificación de los datos y las actividades.



En el caso particular de nuestro proyecto, no se siguió al pie de la letra esta metodología, sino que se tomaron algunas ideas propuestas en ella, y se aplicaron a lo que ya se había hecho utilizando la metodología de [Lar99]. En particular se aplicó el concepto de caso de uso del negocio para describir los procesos del negocio de nuestro caso de estudio y separarlos de lo que serían los casos propios del sistema.

También se complementaron las ideas relativas a la identificación de casos de uso con el trabajo propuesto en [Cer03], en el cual se proporcionan algunas pautas adicionales para realizar esta actividad.

3.3. Necesidades relevadas

En síntesis, se pretende construir un sistema de gestión que cubra todos los aspectos relacionados con esa actividad, los cuales pueden descomponerse en 5 módulos principales:

- Compras
- Ventas
- Caja
- Banco
- Stock

El nuevo sistema sustituiría un conjunto de aplicaciones escritas en Clipper principalmente, que hoy por hoy funcionan de manera independiente, y que se fueron desarrollando a medida que se necesitaban nuevas funcionalidades, para cubrir los aspectos relacionados con la gestión de la empresa. Este tipo de entornos ocasionan una consecuente duplicación de esfuerzo y redundancia en la información, lo cual complica el mantenimiento general.

El interés de ISA en desarrollar un sistema propio y no adquirir uno existente, viene dado por un conjunto de necesidades de la empresa, las cuales se describen a continuación:

- el sistema debe ser completamente accesible vía Web
- se debe poder integrar con sistemas existentes en la empresa
- se desean mantener ciertas características del diseño de los sistemas desarrollados anteriormente

3.4. Resultados obtenidos

Como resultado de esta etapa se obtuvieron los documentos que constituyen el análisis básico del sistema. Dichos documentos son los siguientes:

- Especificación de Requerimientos
- Modelo de Casos de Uso
- Modelo Conceptual

Los mismos podrán encontrarse en los Anexos Particulares de este informe.

3.5. Evaluación

El análisis realizado en esta etapa corresponde a un análisis inicial, que probablemente deba ser refinado más adelante.

Se pueden realizar dos puntualizaciones sobre los requerimientos que se recabaron:

- La primera, es que más allá de que la implementación de este tipo de sistemas sea un problema bastante conocido, este caso presenta algunas particularidades ya mencionadas, como por ejemplo el requisito de seguir el modelo utilizado por los sistemas a sustituir. Esto implicó la tarea adicional de realizar un estudio del funcionamiento de dichos sistemas.
- El otro punto que interesa destacar es que la cantidad de requerimientos analizados fue mayor a la que se preveía inicialmente, con lo cual el tiempo dedicado al análisis también fue mayor al previsto.

----- ¤ -----

Capítulo 4. Alternativas de Diseño

En este capítulo se describe brevemente la metodología de diseño elegida en su forma original, se realiza la correspondencia con las construcciones J2EE planteando el impacto que causa la utilización de Struts en cada una de las capas de la arquitectura lógica y se plantean alternativas de diseño y/o implementación que permiten salvar esas diferencias.

En el capítulo 5 se indicará cuál de las alternativas, planteadas en este capítulo, se ha elegido para el diseño e implementación del sistema de gestión.

4.1. Introducción

La metodología de análisis y diseño orientados a objetos utilizada hasta el momento ([Lar99]) propone la separación entre el modelo, la vista y la lógica de negocio, tal cual lo define el patrón de diseño modelo/vista/controlador utilizado en J2EE. Refleja además, una arquitectura de n capas: presentación, lógica de aplicación - objetos del domino del problema, lógica de aplicación – objetos de servicio, y almacenamiento, mencionando que el análisis y diseño orientados a objetos son más útiles para modelar los niveles lógicos de la aplicación.

El problema es que al pensar en el seguimiento de dicha metodología para el diseño del sistema no resultaba claro la forma en que podría realizarse la implementación de los distintos componentes J2EE a partir de los artefactos de diseño que se obtuvieran.

Se buscó entonces una metodología que estuviera más orientada a la construcción de aplicaciones cuya arquitectura esté basada en el uso de componentes.

Se consideró interesante la incorporación de algunos de los conceptos planteados por la metodología de desarrollo basado en componentes (CBD) propuesta en el reporte técnico [VP03].

Al analizar la posibilidad de su utilización se encontraron algunas diferencias entre el funcionamiento de Struts (framework elegido para implementar la capa web) y las responsabilidades propuestas por la metodología para cada capa de la arquitectura lógica de la aplicación. Se plantearon, entonces, distintas alternativas para alinear el uso de la metodología y Struts.

4.2. Metodología de desarrollo basado en componentes

Esta metodología se basa en casos de uso y está centrada en la definición de una arquitectura que permita observar y manejar globalmente los cambios. Está orientada a aplicaciones empresariales de gran porte.

El paradigma de componentes se focaliza en buscar construir para el cambio; el objetivo de un componente es que sea fácilmente reemplazable por otro.

Recordemos que la arquitectura J2EE es basada en componentes y por lo tanto la utilización de una metodología orientada a ello puede ayudar a lograr un mejor diseño.

***Nota**: Es recomendable la lectura del reporte [VP03] para facilitar la comprensión de los conceptos manejados en esta sección*.

4.2.1. Metodología

La metodología de desarrollo abarca las tres primeras fases del proceso de desarrollo de software propuesto por Rational Software (RUP – Rational Unified Process) y plantea la realización de actividades adicionales en alguna de las fases. Las fases abarcadas son las siguientes:

1. **Inception:** Esta etapa corresponde al conocimiento del negocio, destacándose las actividades de identificación y clasificación de sus procesos. El workflow de las actividades a realizar se presenta en la siguiente figura:

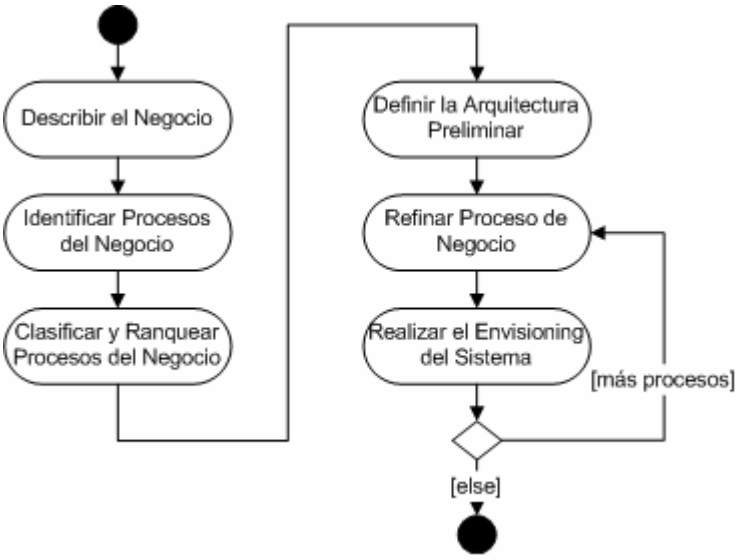


Figura 5 - CBD: Workflow de la fase Inception

2. **Elaboration:** En esta fase se atacan los procesos de negocio más críticos para la arquitectura, en el orden establecido por el ranqueo de los procesos realizado en la etapa anterior. Las actividades incluidas en esta etapa son las que se muestran en la siguiente figura:

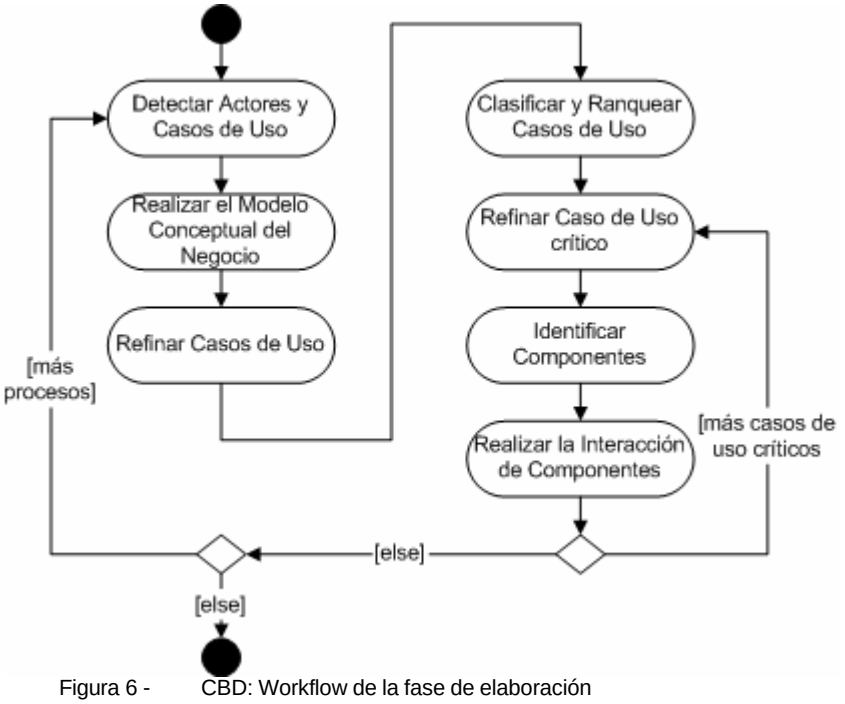


Figura 6 - CBD: Workflow de la fase de elaboración

Las últimas dos actividades son agregadas por la metodología CBD a la propuesta RUP:

- Identificar Componentes: actividad utilizada para lograr la presentación de interfaces y especificaciones de componentes en una arquitectura de componentes inicial. Las tareas que deben realizarse son las siguientes:

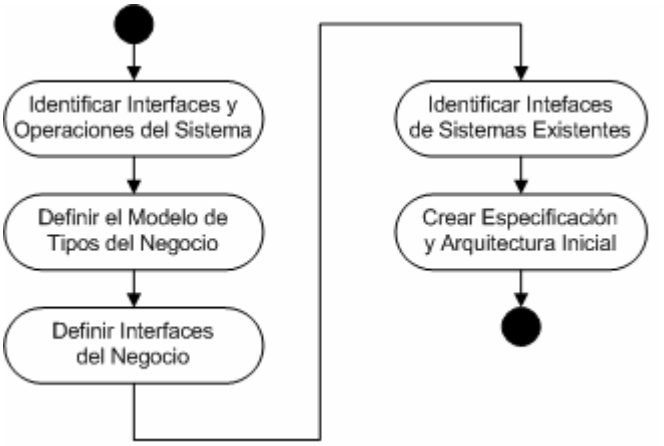


Figura 7 - CBD: Tareas de la actividad Identificar Componentes

- Realizar la Interacción de Componentes: en esta actividad se decide la forma en que trabajarán juntos los componentes detectados en la actividad anterior para lograr el cumplimiento de las funcionalidades deseadas. Las tareas que deberán realizarse son las siguientes:

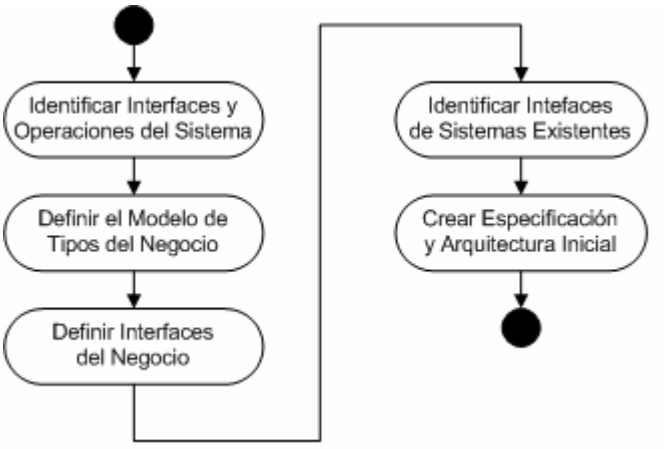


Figura 8 - CBD: Tareas de la actividad Realizar la Interacción de Componentes

3. **Construction:** Las actividades de esta etapa son análogas a las de la etapa anterior, pero como se tiene una arquitectura suficientemente estable, la metodología CBD agrega una nueva actividad denominada Especificación de Componentes al final del workflow. El objetivo de esta actividad es que se definan las vistas abstractas de la información manejada por cada componente y se especifiquen formalmente las operaciones de las interfaces. Las tareas que deben realizarse para llevarla adelante son las siguientes:

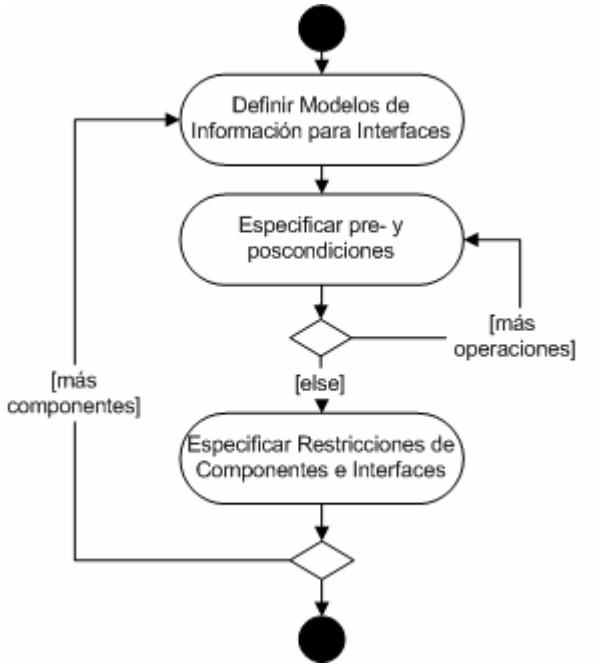


Figura 9 - CBD: Tareas de la actividad Especificación de Componentes

4.2.2. Arquitectura

Se plantea una arquitectura de múltiples capas en la cual cada capa sugiere un tipo diferente de componente, e indica el rol que juegan los componentes que residan en ella.

La arquitectura propuesta es la siguiente:

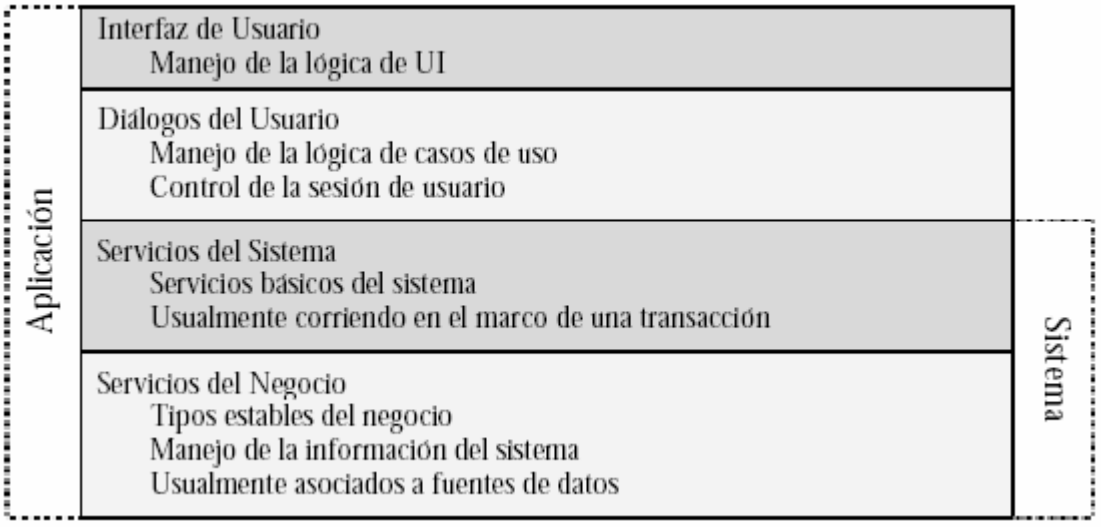


Figura 10 - Arquitectura CBD

Esta definición de arquitectura cubre aspectos únicamente lógicos y es totalmente independiente de la tecnología con la que se realice la implementación de los componentes.

A continuación se describe brevemente cada una de estas capas:

- **Capa de Interfaz de Usuario:** Se encarga de la presentación alusuario. Como ejemplo puede contener formularios, páginas Web, etc.

- **Capa Diálogos de Usuario:** Cada caso de uso implica un "diálogo" con el usuario. En esta capa se definen componentes denominados *Diálogos de Usuario*, en los que se encapsula la lógica del diálogo de cada caso de uso. Los diálogos mantienen su estado y conocen además qué servicios requerir a la capa inferior para llevar a cabo cada caso de uso.

- **Capa de Servicios del Sistema:** Exporta operaciones que resuelven requerimientos del sistema. Está organizada en subsistemas. Contiene además adaptadores a sistemas externos.

- **Capa de Servicios del Negocio:** Contiene componentes que corresponden a tipos estables del negocio. Administra la información persistente del negocio.

4.3. Correspondencia con la plataforma J2EE y Struts

Como ya se mencionó, la metodología planteada por [VP03] se basa en la realización del diseño en forma independiente de la tecnología que va a utilizarse para implementarlo. Una vez definidos los componentes residentes en cada capa de la arquitectura lógica, es necesario realizar el mapeo con las construcciones de la tecnología particular a utilizar (J2EE en este caso), de forma tal que dicho mapeo se pueda implementar en forma directa.

En [PRV03] se proporciona una correspondencia entre la arquitectura lógica propuesta en [VP03] y J2EE, agregando previamente a la arquitectura inicial la capa *UI Diálogos de Usuario*. Dicha capa será la encargada de coordinar la presentación al usuario.

El mapeo obtenido a nivel general es el siguiente:

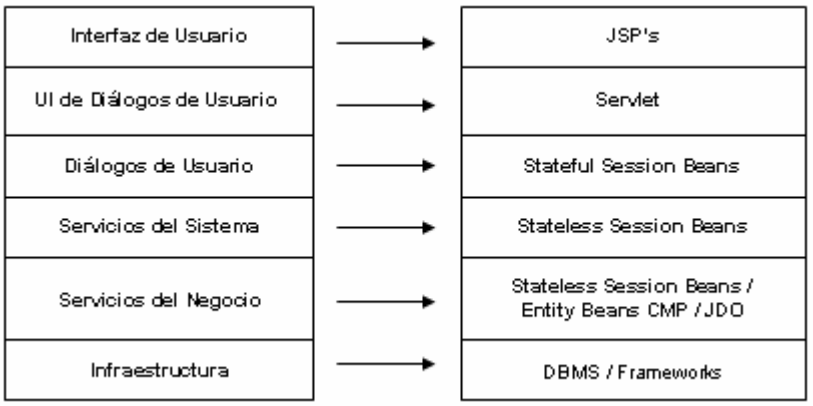


Figura 11 - Arquitectura lógica propuesta y mapeo con J2EE

Recordemos que en nuestro caso particular utilizaremos J2EE junto con Struts. En el reporte técnico [BFF03], se menciona que el uso del framework Struts, no es del todo compatible con la arquitectura propuesta por esta metodología.

El problema al que se refiere dicho trabajo, viene dado por la manera en que los componentes de Struts estimulan a implementar la lógica del diálogo con el usuario (es decir, la lógica que controla la secuencia de páginas mostradas al usuario).

Al igual que en el mapeo propuesto en la figura anterior, Struts utiliza un Servlet para controlar y direccionar las invocaciones efectuadas por los clientes. La diferencia radica en que Struts permite definir *clases de acción* (Action Classes), que son invocadas para atender cada solicitud.

Por lo general, los ejemplos de implementaciones con Struts incluyen la lógica de qué página mostrar directamente en las clases de acción, lo cual hace que esa lógica se distribuya entre dichas clases y no se concentre en un único componente para cada caso de uso, como propone la metodología con los Diálogos de Usuario.

En este trabajo proponemos dos alternativas para alinear el uso de Struts y la metodología basada en componentes. Dichas alternativas se reflejarán en las capas donde sea necesaria una discusión entre ambos enfoques, ya que existen capas cuyo mapeo a J2EE es análogo para ambos.

4.3.1. Interfaz de Usuario

Para esta capa se propone una organización con el estilo cliente-servidor, utilizando en el cliente un navegador de Internet para acceder a páginas Web basadas en HTML y HTML forms, y un servidor de páginas Web dinámicas en el lado del servidor, empleando para ello la tecnología JavaServer Pages (JSPs).

En este caso, el uso de Struts no interfiere con el mapeo propuesto ya que también propone el uso de HTML para el cliente y JSPs para el servidor.

Adicionalmente Struts proporciona soporte para un conjunto de funcionalidades, que facilitan el desarrollo de la Vista de la aplicación, como ser una librería de etiquetas JSP y el uso de ActionForms, que permiten realizar chequeos sobre la consistencia de los datos ingresados por el usuario.

4.3.2. UI de Diálogos de Usuario y Diálogos de Usuario

En la correspondencia propuesta originalmente ([PRV03]) estas dos capas son analizadas por separado, pero en este caso se analizarán de forma conjunta debido a que el uso de Struts puede llevar a que se visualicen como una única capa en la implementación.

La metodología utilizada ([VP03]) propone las siguientes responsabilidades para estas capas:

- La capa UI Diálogos de Usuario es la capa controlador. Se encarga de recibir la solicitud HTTP realizada por el cliente e invocar al Diálogo de Usuario que corresponda, pero no conoce qué operación debe invocarse dentro de ese diálogo.
- En la capa de Diálogos de Usuario se define un Diálogo de Usuario para cada caso de uso, que es utilizado para saber qué servicios y en qué orden requerir a la capa inferior, así como para determinar la próxima página a visualizar.

El mapeo que se propone en [PRV03] para estas capas es el siguiente:

- Se utiliza un servlet para la capa UI que encapsule la información recibida de los HTML Forms en un JavaBean, la envíe al Diálogo de Usuario correspondiente, y devuelva al usuario una nueva JSP que muestre el resultado obtenido. Los HTML Forms tienen seteado el atributo *action* de forma tal que siempre invoquen al servlet controlador. El Diálogo de Usuario que el servlet debe invocar vendrá especificado en el objeto sesión.
- Los Diálogos de Usuario se implementan como Stateful SessionBeans utilizando el patrón de diseño *Command.* Para ello se define una interfaz IDialog con una única operación *execute*, y se utilizan JavaBeans para el intercambio de datos entre los diálogos y la capa superior. Los datos de la solicitud se encapsulan en un JavaBean de tipo Data y los datos resultantes de la ejecución de la operación se devuelven en un JavaBean de tipo Result, el cual contiene un objeto Data y el estado en que quedó el diálogo. El estado es utilizado por el servlet para determinar la siguiente página a mostrar.

A continuación se analizan los diagramas de secuencia del mapeo explicado en el párrafo anterior, y del funcionamiento general de Struts, explicado en la sección 2.5.2.

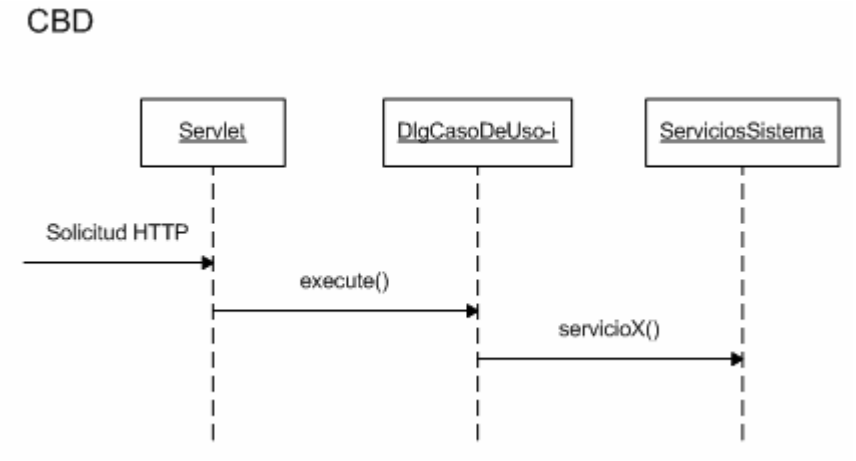


Figura 12 - Diagrama de secuencia CBD

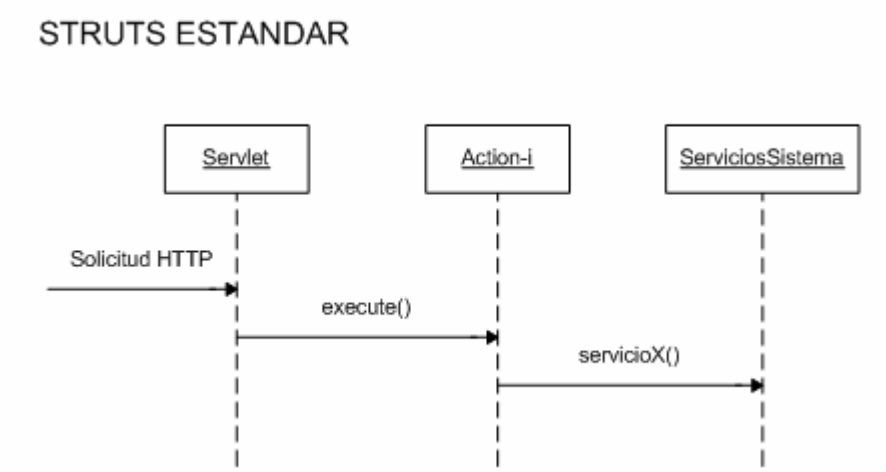


Figura 13 - Diagrama de secuencia Struts Estándar

Nota: Por simplicidad se omite el componente Request Processor de Struts, que interviene entre el Servlet y los objetos Action.

Si bien los diagramas pueden resultar similares, existen algunas diferencias entre estos dos enfoques que resumimos a continuación:

CBD	Struts
La lógica de la máquina de estados correspondiente a cada caso de uso se encapsula dentro de los Diálogos de Usuario, liberando al controlador del conocimiento de dicha lógica.	La lógica para la secuencia de operaciones de un caso de uso se encuentra implícita en la capa controlador. Cada Action invocada por el servlet se corresponde con una operación.
La implementación de los componentes command del Command Pattern, que en este caso serían los Diálogos de Usuario, es realizada mediante EJBs de sesión con manejo de estado.	Los componentes command se implementan como clases que extienden la clase Action provista por Struts. En este caso se está usando una clase Java común y no un EJB, con lo cual se pierden las prestaciones que brinda ese tipo de componente.
Los datos enviados por el controlador a los comandos (Diálogos de Usuario) se encapsulan en JavaBeans de tipo Data. Los datos devueltos por los comandos se encapsulan en JavaBeans de tipo Result, que contienen un objeto Data con datos de respuesta y un estado que determina la siguiente página a	Los datos enviados por el controlador a los comandos (clases Action), se encapsulan en JavaBeans de tipo ActionForm. Los comandos devuelven objetos ActionForward, que indican la siguiente página a mostrar.

mostrar.	
El atributo *action* de los HTML forms está seteado de forma tal que se ejecute el servlet controlador al hacer el submit. El diálogo que debe ejecutarse va seteado en el objeto sesión.	El atributo *action* de los HTML forms está seteado con el nombre de la acción que debe ejecutarse. El ActionServlet captura dicha solicitud porque el servidor está configurado para que dicho servlet reciba todas las solicitudes.

Las alternativas que proponemos para lograr una alineación entre ambos enfoques son las siguientes:

4.3.2.1. Alternativa 1: Modificación de la arquitectura planteada por la metodología

Ya que Struts incluye en sus componentes de control el servlet controlador y las acciones que deben realizarse, podría pensarse en unir las capas de UI Diálogos de Usuario y la de Diálogos de Usuario planteadas por la metodología CBD en una única capa que contenga el controlador y la máquina de estados.

La nueva arquitectura y su mapeo con J2EE sería el siguiente:

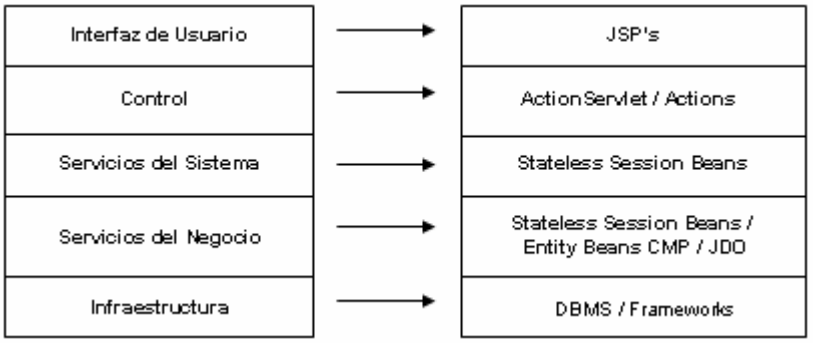


Figura 14 - Nueva arquitectura lógica y mapeo Struts

4.3.2.2. Alternativa 2: Implementación con Struts adaptada a la metodología

Otra posibilidad es que los componentes de Struts sean utilizados de forma tal que la implementación pueda corresponderse con la arquitectura lógica definida en la metodología.

El mapeo propuesto para este caso sería el siguiente:

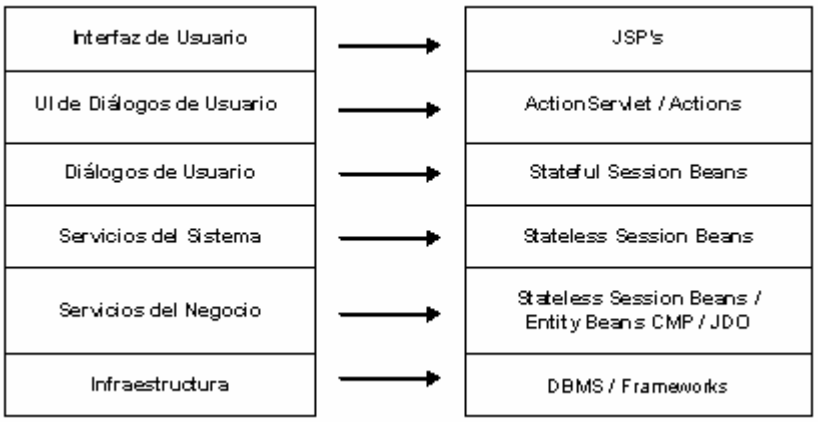


Figura 15 - Arquitectura lógica CBD y mapeo Struts propuesto

Para lograr esta correspondencia las clases Action ubicadas en la capa de UI Diálogos de Usuario, junto con el ActionServlet, se encargarían de llamar al diálogo que corresponda y no contendrían la lógica de las operaciones ellas mismas. Luego el diálogo será quien conozca la máquina de estados y se encargue de ejecutar la operación que corresponda, tal como se propone en el mapeo de la metodología CBD.

El nuevo diagrama de secuencia sería el siguiente:

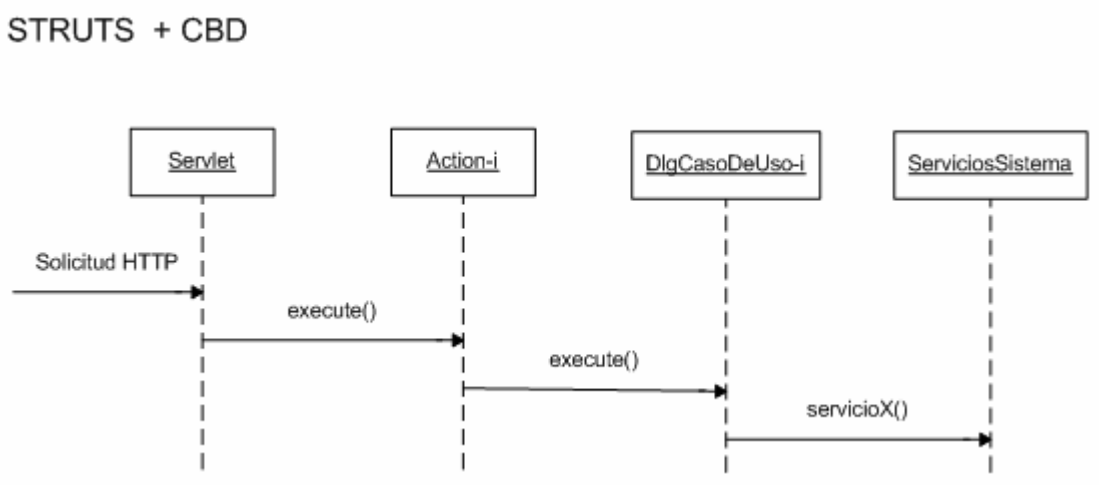


Figura 16 - Diagrama de secuencia Struts & CBD

En este caso las clases Action funcionarían como adaptadores (Adapter Pattern) para los datos intercambiados entre el ActionServlet y los Diálogos de Usuario.

Se tendría una clase Action por cada Diálogo de Usuario. La idea es que un mismo objeto Action siempre invoque a un mismo Diálogo de Usuario.

Luego, todas las acciones correspondientes a un mismo caso de uso, se asocian en el archivo XML de Struts a la clase Action encargada de invocar al diálogo correspondiente a ese caso de uso. Cada acción utilizará su propio ActionForm para el pasaje de los datos a la clase Action, la cual deberá pasar esos datos en un objeto Data al diálogo correspondiente.

4.3.3. Servicios del Sistema

Al aplicar la metodología de desarrollo basado en componentes a este nivel, se crea una interfaz de sistema por cada caso de uso detectado y se define un componente de sistema por cada proceso de negocio. Dicho componente será el encargado de realizar todas las interfaces correspondientes a un mismo proceso.

Los componentes de esta capa no necesitan mantener su estado, ya que el mismo es mantenido por la capa superior. Por lo tanto podremos utilizar Stateless SessionBean.

4.3.4. Servicios del Negocio

De acuerdo a la metodología CBD, los componentes de esta capa se detectan a partir del modelo de tipos de negocio. Se define como un Core Type a cada tipo cuya existencia sea independiente dentro del negocio. Para cada uno de estos tipos se crea una interfaz de negocio responsable de manejar todas sus instancias, la cual se identifica como el manager del core type.

La implementación de cada manager se realiza utilizando un EJB de tipo Stateless SessionBean. Cada componente que necesite servicios del manager deberá solicitar una instancia del mismo.

Es necesario definir la forma en que los managers van a administrar los datos que se encuentran en distintas fuentes (data sources).

Para el caso de objetos de negocio persistentes podemos plantearnos las siguientes dos alternativas extremas para manejarlos:

4.3.4.1. Alternativa 1: Persistencia manejada con clases Java y JDBC

Una opción posible es la de definir los objetos de negocio utilizando clases Java planas y realizar llamadas JDBC (generalmente vía JNDI) en los métodos donde sea necesario almacenar u obtener datos de una base de datos [Per02].

Para manejar JDBC mediante llamadas directas existen las siguientes opciones ([Per02] [Jak03]):

Como regla, siempre debe utilizarse un pool de conexiones para acceder a una base de datos. A pesar de que la conexión inicial es una operación costosa, la creación del pool permite que varias aplicaciones puedan utilizar las conexiones ya existentes evitando que deban establecerse nuevamente, lo cual mejora los tiempos de respuesta [WSAD03].

- Si el contenedor utilizado proporciona un pool de conexiones que implementa la interfaz *javax.sql.DataSource* y además es configurable desde las propiedades que ofrece al usuario, se puede obtener la conexión a la base de datos a través de él. (Particularmente, el contenedor utilizado en este proyecto, WebSphere Application Server, lo provee).

- En caso de que el contenedor utilizado no brinde la posibilidad de crear un pool de conexiones, puede integrarse un componente ofrecido por algún otro proveedor.

  Jakarta, por ejemplo, dispone de la clase BasicDataSource que forma parte del proyecto The Jakarta Commons dbcp's BasicDataSource: org.apache.commons.dbcp.BasicDataSource; este componente DBCP (Database Connection Pool) puede bajarse desde http://jakarta.apache.org/commons/dbcp/ downloads.html e incorporarlo a la aplicación.

4.3.4.2. Alternativa 2: Persistencia manejada por EJBs

Otra opción es utilizar la tecnología que nos ofrece J2EE para representar objetos de negocio (EJBs) y manejar EntityBeans para los objetos persistentes.

¿CMP o BMP?
Al utilizar EJBs la persistencia puede ser manejada por el contenedor (Container Managed Persistence) o también puede utilizarse la opción de que sea manejada por el bean (Bean Managed Persistence).

En el caso de los CMPs el contenedor es el responsable de mantener el estado y manejar todos los aspectos de acceso a ellos, permitiendo que el programador se concentre en la lógica de negocio. Solo se necesita que el programador especifique qué campos manejará el contenedor. El servidor de aplicaciones provee servicios de cache, estrategias de lockeo de base de datos y escalabilidad.

Utilizando BMP, el programador es quien debe encargarse de escribir el código para manejar la persistencia, al igual que en la alternativa de utilización de clases Java y JDBC descripta en el punto anterior.

Entonces, podría pensarse que la mejor opción es utilizar CMP y delegar todo el manejo de la persistencia en el contenedor.

El problema es que hay que tener en cuenta los requerimientos no funcionales de performance de la aplicación [BFF03]. Cuando se realicen operaciones que afectan a varios EntityBeans, el uso de CMP's puede ser muy ineficiente [PRV03].

4.3.4.3. ¿Qué alternativa elegir?

En [BFF03] se realizó una comparación entre el uso de JDBC y EJBs de tipo CMP. Las ventajas y desventajas que pueden extraerse de dicho artículo para cada caso son las siguientes:

	EJB's CMP		**JDBC**	
	Ventajas	**Desventajas**	**Ventajas**	**Desventajas**
Lenguaje de consulta hacia la base de datos	La utilización del lenguaje EJB QL permite realizar consultas en forma independiente al lenguaje de consulta de la base de datos.	No pueden utilizarse comparaciones ni restricciones para campos de tipo fecha. Esto lleva a obtener más resultados que los deseados para determinadas consultas y a tener que realizar un filtro posterior por fecha utilizando lógica Java.	Pueden realizarse consultas con filtro por fechas.	Debe utilizarse SQL (el cual difiere de una base de datos a otra) o alguna solución propietaria del servidor utilizado.
Tiempos de respuesta		El promedio en segundos del tiempo de retorno de resultados para las consultas realizadas fue muy superior al obtenido en el caso de la utilización de JDBC.		
Tiempo de implementación	El desarrollo de CMP puede ser un método eficiente si los beans son simples y si se tiene experiencia en el tema. De todas formas, no se detectaron grandes diferencias de tiempo, excepto en algunos casos puntuales.			

Como se puede ver, la única ventaja de los CMP's es la abstracción del lenguaje de consulta [BFF03]. En dicho artículo se ha planteado como un trabajo futuro el análisis de las ventajas que puede ofrecer la utilización del caché de objetos provista por J2EE (útil para datos consultados frecuentemente ya que se evitaría la realización de la consulta nuevamente y podría reducir los tiempos de respuesta).

De acuerdo a la complejidad de los beans de la aplicación y a los requerimientos de performance habrá que analizar qué alternativa de manejo de persistencia es la más adecuada.

Patrón recomendado: JDBC for Reading

En el libro EJB Design Patterns ([Mar02]) se describen una serie de patrones que ayudan a lograr aplicaciones basadas en EJBs de mejor calidad. Para el caso de la persistencia de datos se recomienda el uso del patrón "JDBC for reading", el cual se describe a continuación:

Un caso muy común en las aplicaciones Web es la necesidad de presentar una tabla con una larga lista de datos a los efectos de su lectura y nada más. Estos datos serán levantados de la base de datos y presentados al usuario.

Desventajas de los EJBs al recuperar datos

- Utilizando BMP's, una simple operación de consulta que devuelva N datos requiere N + 1 llamadas a la base de datos. Esto también sucede en algunas implementaciones de CMP's; no en todas porque algunos contenedores proveen una buena implementación de estos beans y logran una mejor performance.
  La situación es la siguiente: para leer datos de N EntityBeans, se debe realizar una primer llamada al método de búsqueda (un acceso a la base de datos), luego el contenedor va a ejecutar el método ejbLoad() por cada EntityBean devuelto por el método de búsqueda, lo cual implica N llamadas más.
  Cada una de estas llamadas genera un lock de la conexión a la base en el pool de conexiones, se abren y cierran conexiones, etc. Incluso, si la base de datos está en otro servidor, cada llamada aumenta el tránsito en la red.
- Si las interfaces de los beans son remotas aumenta aún más la sobrecarga.
- Con operaciones simples que impliquen joins entre varias tablas la performance del sistema baja significativamente. Utilizando tanto BMP, como CMP, esta situación implica la instanciación de múltiples EntityBeans, o sea muchas llamadas a bases de datos y llamadas remotas; además del overhead que se produce en el servidor al atravesar las relaciones entre los múltiples EntityBeans.

Ventajas del uso de JDBC para recuperar datos

Además de no producirse los problemas mencionados anteriormente, el uso de JDBC tiene algunas otras ventajas:

- En operaciones simples de consulta se evita la sobrecarga por el manejo transaccional, ya que no es necesario manejar transacciones al realizar este tipo de operaciones. Utilizando la opción JDBC con un Stateless SessionBean se pueden deshabilitar; en cambio en el caso de EntityBeans es imposible utilizarlos sin transacciones.
- Se puede utilizar el caché que maneje la propia base de datos, la cual puede mantener el resultado final de una consulta y no como en el caso de los EntityBeans que mantienen un caché por cada tabla individual (cada EntityBean el suyo).
- Se pueden obtener los datos que realmente se necesitan. Al utilizar JDBC se seleccionan las columnas exactas que interesan y de cualquier cantidad de tablas. En el caso de los EntityBeans, cargan todos los atributos del bean, aunque el cliente quizás necesite solo uno.
- La lectura entera de una tabla puede realizarse en una única "gran" llamada, lo cual contrasta con las N + 1 llamadas que necesitan los EntityBeans, más las llamadas que necesiten realizar a sus interfaces remotas.

Por estas razones, en el caso de consulta de datos, los beneficios de los EntityBeans son poco claros [Mar02].

Utilizando interfaces locales y una buena implementación de CMP se pueden reducir los problemas de performance que éstos presentan.

Al utilizar BMP, la única forma de disminuir estos problemas es habilitando el caché de EntityBeans, el cual solo puede utilizarse si la base de datos no es modificada por fuera de la aplicación EJB y si el servidor de aplicaciones no está en cluster con otro servidor.

El patrón "JDBC for reading" dice que al realizar operaciones de consulta de datos sobre bases de datos relacionales es recomendable utilizar JDBC. Es más rápido y más eficiente.
Los EntityBeans tienen ventajas en la realización de operaciones de actualización, no de consulta.

Como al utilizar JDBC es el programador quien genera la lógica de negocio y la lógica de persistencia, puede pasar que queden acopladas. Además puede ser que la aplicación sea más propensa a tener bugs y que se vuelva menos mantenible. Para evitarlo se recomienda utilizar el patrón Data Access Object (DAO), descripto en el anexo correspondiente a patrones de diseño J2EE, el cual permite desacoplar el acceso a datos de la lógica de negocio, creando un objeto intermedio que se encargue del acceso a la base de datos que corresponda.

Utilización de EJBs para actualización de datos

Los conceptos de integridad de los datos de objetos de negocio y sus relaciones, que no son importantes al listar tablas de datos, sí lo son a la hora de realizar actualizaciones sobre la base de datos.

Los EntityBeans encapsulan tanto los datos como las reglas para modificarlos. Cuando se actualiza un atributo en un EntityBean, puede necesitarse que se realice una validación lógica sobre esos cambios y que se deban generar cambios sobre otros EntityBeans de la aplicación. Estas situaciones son manejadas automáticamente.

Si en este caso se utilizara JDBC el programador es quien debe escribir todo el código para enlazar la lógica de negocio con la complejidad de la lógica de datos; todas las reglas, relaciones y validaciones requeridas por cada concepto particular del negocio. El sistema será muy frágil a los cambios de los requerimientos de negocio de la aplicación.

4.4. Evaluación

Si bien la utilización de frameworks en la capa web de la aplicación tiene las ventajas descriptas en el punto 2.4.2, es importante que el diseño de la aplicación sea realizado de tal forma que se mantenga una clara separación entre las distintas capas de la arquitectura y que los componentes desarrollados no pierdan la característica de ser fácilmente reemplazables por otros. Tal como se menciona en el reporte [VP03], "*Un software de calidad debe estar preparado para absorver todo tipo de cambios con el menor impacto posible*".

Como se ha visto anteriormente, en este trabajo ha sido necesario analizar el impacto de Struts en la arquitectura de la aplicación y, ya que no se adapta a la arquitectura propuesta por la metodología de diseño elegida, plantear alternativas para la realización del diseño e implementación del sistema utilizándolo.

En el próximo capítulo se indicará cuál de las alternativas propuestas se ha utilizado para el diseño e implementación del sistema de gestión y otro tipo de decisiones que ha sido necesario tomar.

----- ¤ -----

Capítulo 5. Implementación de un Caso de Estudio

En este capítulo se fija el alcance del caso de estudio abordado en este proyecto y se describen las decisiones tomadas para sortear las dificultades encontradas al momento de realizar su implementación.

Uno de los puntos centrales de este trabajo, consistió en investigar la posibilidad de utilizar el framework Struts para implementar las capas superiores de la arquitectura propuesta por la metodología CBD utilizada. Por este motivo, se hace un mayor énfasis en el análisis de las capas *UI Diálogos de Usuario* y *Diálogos de Usuario*, y no se profundiza tanto en las capas inferiores.

Para esta etapa se tomó como referencia el trabajo realizado en [BFF03], el cual propone ideas interesantes para resolver algunas cuestiones propias de la implementación. También se aplicaron ideas nuevas para la resolución de algunos detalles que se dejan planteados en dicho trabajo, como ser la manera en que se maneja la composición de casos de uso.

5.1. Alcance

Luego de la obtención de todos los casos de uso del sistema de gestión se decidió, junto con los integrantes del área de gestión de la empresa, que los casos a diseñar e implementar en este proyecto serían el caso de uso "Registrar Venta" y el caso de uso "Registrar Cobranza Factura Crédito" realizándoles algunas simplificaciones, de forma tal que funcionen como se describe a continuación:

Caso de Uso Nº 8	**Registrar Venta**
Actores	Administrativo
Tipo	Primario
Descripción	El administrativo selecciona el cliente al que se realiza la venta, ingresa los datos de los productos vendidos e indica, entre otras cosas, si es una venta contado o crédito, la moneda y la forma de pago. El administrativo finaliza el ingreso de datos. El sistema calcula el total y registra la venta.

Caso de Uso Nº 16	**Registrar Cobranza Factura Crédito**
Actores	Administrativo
Tipo	Primario
Descripción	El administrativo indica el cliente que va a realizar el pago, la forma de pago y la moneda, entre otros datos. El sistema le permite elegir una o más facturas de las que debe dicho cliente. El administrativo finaliza el ingreso de datos. El sistema registra la cobranza.

Obs. En el documento de casos de uso se encuentra la descripción de alto nivel de los casos de uso originales, junto con las referencias de los requerimientos relacionados con cada caso de uso.

Dado que estos dos casos tienen en común la actividad de seleccionar un cliente, se creó un nuevo caso de uso para encapsular dicha tarea, el cual es incluido por los dos casos iniciales.

Caso de Uso Nº 8.1	**Seleccionar Cliente**
Actores	Administrativo
Tipo	Primario
Descripción	El administrativo selecciona el cliente vinculado a la operación comercial que realizará en el sistema

Analizando además los casos de uso generales para determinar el impacto que podría tener cada uno de ellos en la arquitectura, se vio la necesidad de separar el caso de uso "Configurar el Sistema" en dos

nuevos casos e implementar uno de ellos. El caso original (que puede verse en los anexos), incluye las actividades de gerenciamiento de información (carga de tablas maestras), las cuales pueden ser postergadas hasta la etapa de construcción, y además tiene incluida la tarea de definición de transacciones y operaciones, la cual es un requerimiento de diseño para el sistema.

Tanto los integrantes del área de gestión de la empresa, como el responsable del proyecto por parte de la misma, desean mantener una característica de diseño presente en uno de los sistemas utilizados en la actualidad.

Este requerimiento implica que se pueda configurar el comportamiento de las operaciones comerciales que realice el sistema y que no se afecte en forma real cada cuenta, sino que cada operación que se realice debe quedar registrada, y a la hora de realizar informes se realicen los cálculos en función de dichos registros y su definición. Este punto impacta en el diseño de todo el sistema y por tal motivo será extraído hacia un nuevo caso de uso y se abordará desde el inicio.

A continuación se detalla el caso de uso obtenido:

Caso de Uso Nº 2.1	**Configurar Operaciones y Transacciones**
Actores	Administrador del Sistema, Supervisor
Tipo	Primario
Descripción	El administrador del sistema, o el supervisor, configura las transacciones y operaciones comerciales del sistema.

5.1.1. Diagrama de casos de uso

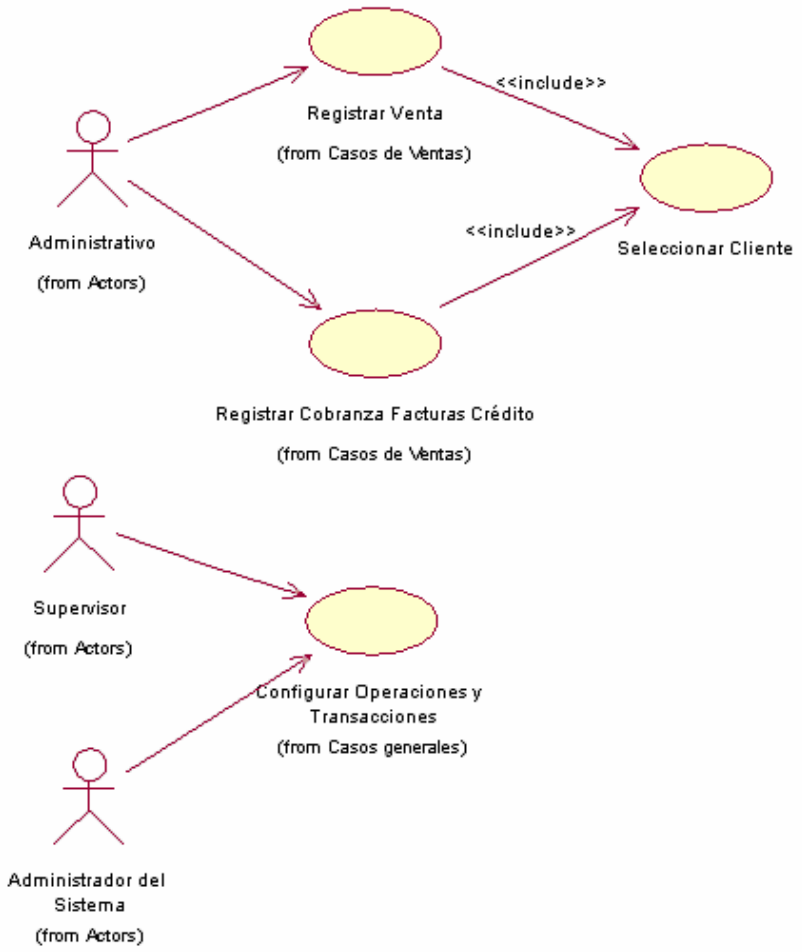


Figura 17 - Diagrama de casos de uso del caso de estudio

En el siguiente capítulo se describirá con detalle el caso de uso “Registrar Venta” y se comentarán los componentes desarrollados en cada capa para dicho caso de uso.

5.1.2. Modelo Conceptual

El modelo conceptual obtenido a partir de los requerimientos analizados para el caso de estudio es el siguiente:

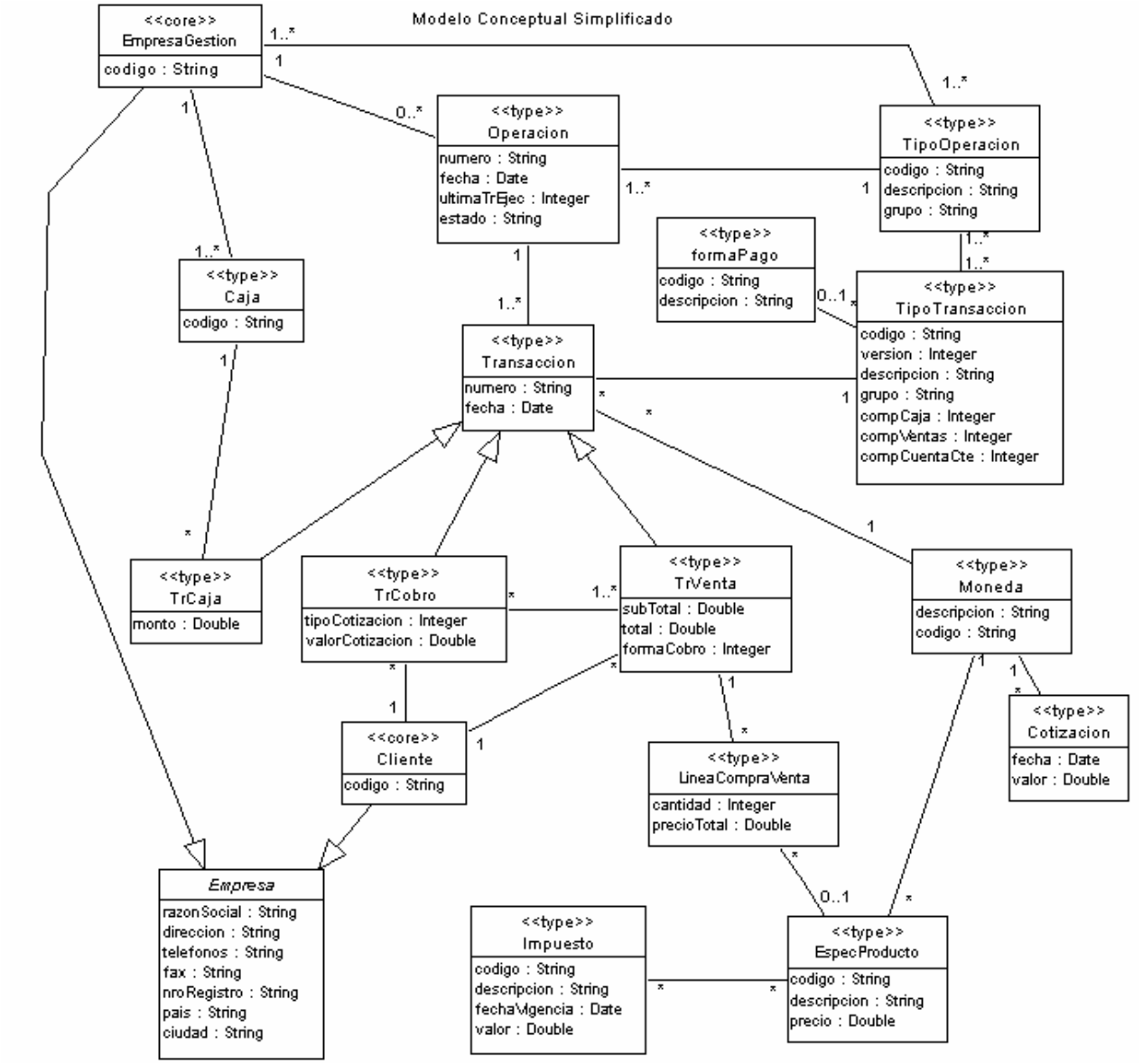


Figura 18 - Modelo Conceptual para el caso de estudio

5.2. Arquitectura lógica

Como se discutió en el capítulo anterior, dado el funcionamiento estándar de Struts se definiría una única capa, que podría llamarse capa de control, la cual contenga el servlet y las acciones que deben invocarse de acuerdo a la solicitud (punto 4.3.2.1). El problema de esta solución es que el diseño queda muy vinculado a la utilización de este framework y si más adelante se decide cambiar por otro framework por ejemplo, será necesario modificarlo.

Por lo tanto, para mantener la independencia del diseño respecto a la utilización del framework Struts o no, se ha elegido la alternativa detallada en el punto 4.3.2.2 (Alternativa 2: Implementación con Struts adaptada a la metodología).

Luego de aplicar algunos conceptos de la metodología CBD y de tomar algunas decisiones adicionales para el diseño del caso de estudio definido en la sección anterior, se obtuvo el diagrama que se muestra a continuación para la arquitectura lógica del sistema:

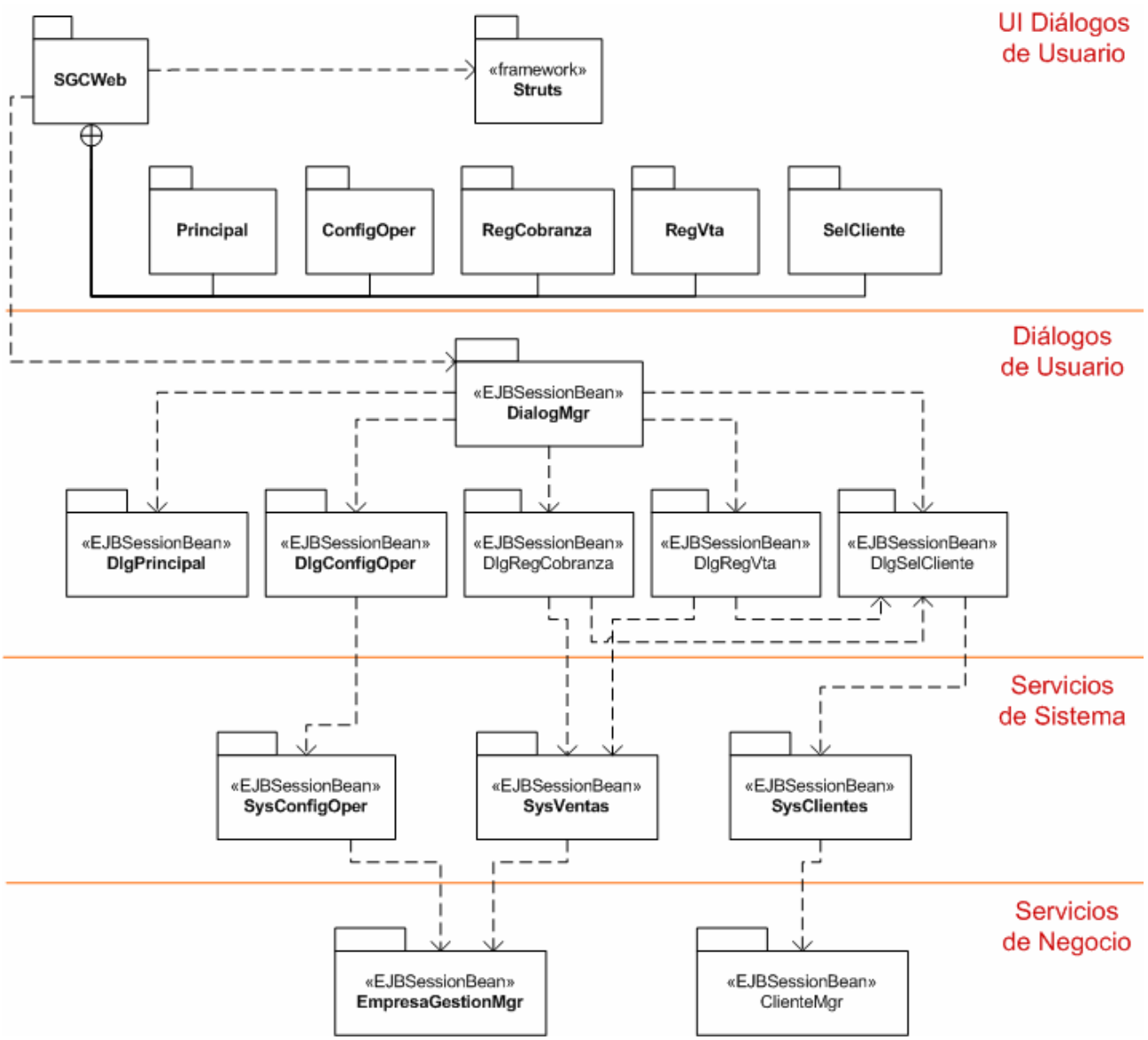


Figura 19 - Arquitectura lógica del caso de estudio

***Nota**: Los detalles de la aplicación de la metodología al caso de estudio pueden encontrarse en los Anexos Particulares de este informe.*

A continuación se comentarán los detalles más relevantes para cada capa:

5.2.1. Capa UI Diálogos de Usuario

Esta capa es la encargada de controlar el flujo de las invocaciones enviadas por los clientes en forma de solicitudes HTTP, y generar las correspondientes respuestas. Además de los componentes básicos de Struts, contiene aquellos componentes particulares a esta implementación, que el programador debe proporcionar al framework. Estos últimos se agrupan en el paquete SGCWeb y sus subpaquetes.

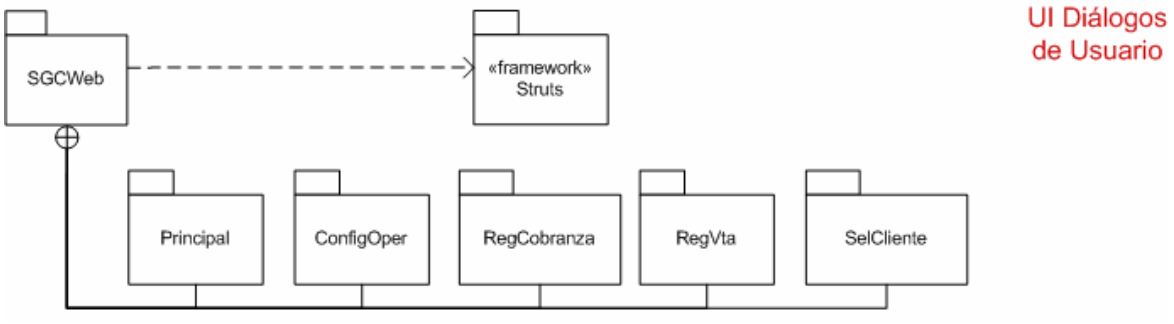


Figura 20 - Capa UI Diálogos de Usuario

Las tareas realizadas para implementar esta capa utilizando el framework Struts fueron las siguientes:

- En la capa Interfaz de Usuario (la cual no será analizada) se implementó una página JSP para cada estado de los casos de uso a realizar.
- En la configuración de Struts, por cada página JSP se definió: un nombre de acción, que es utilizado en las solicitudes enviadas por la página, y un ActionForward, que define un alias para la página, y que se utiliza para referenciarla dentro del código manejado por Struts.
- Se implementó una única clase Action llamada ActionSGC, para manejar las solicitudes de todas las acciones. Cada acción definida está mapeada a esta clase Action.
- Por cada página JSP se implementó una clase ActionForm, para encapsular los datos de la misma. La acción definida para cada página también fue mapeada con el correspondiente ActionForm.

El paquete SGC contiene:

- La clase ActionSGC.
- Una clase denominada FormSGC, que implementa funcionalidades comunes a los ActionForm de cada página.
- Un paquete por cada caso de uso detectado, en el que se incluyen los ActionForm de las páginas correspondientes a ese caso de uso.

***Nota:** El paquete Principal que se observa en el diagrama, corresponde a un caso de uso especial que representa el estado del usuario en el sistema, y que será explicado en la sección correspondiente a la capa Diálogos de Usuario.*

A continuación se muestra el funcionamiento general de esta capa:

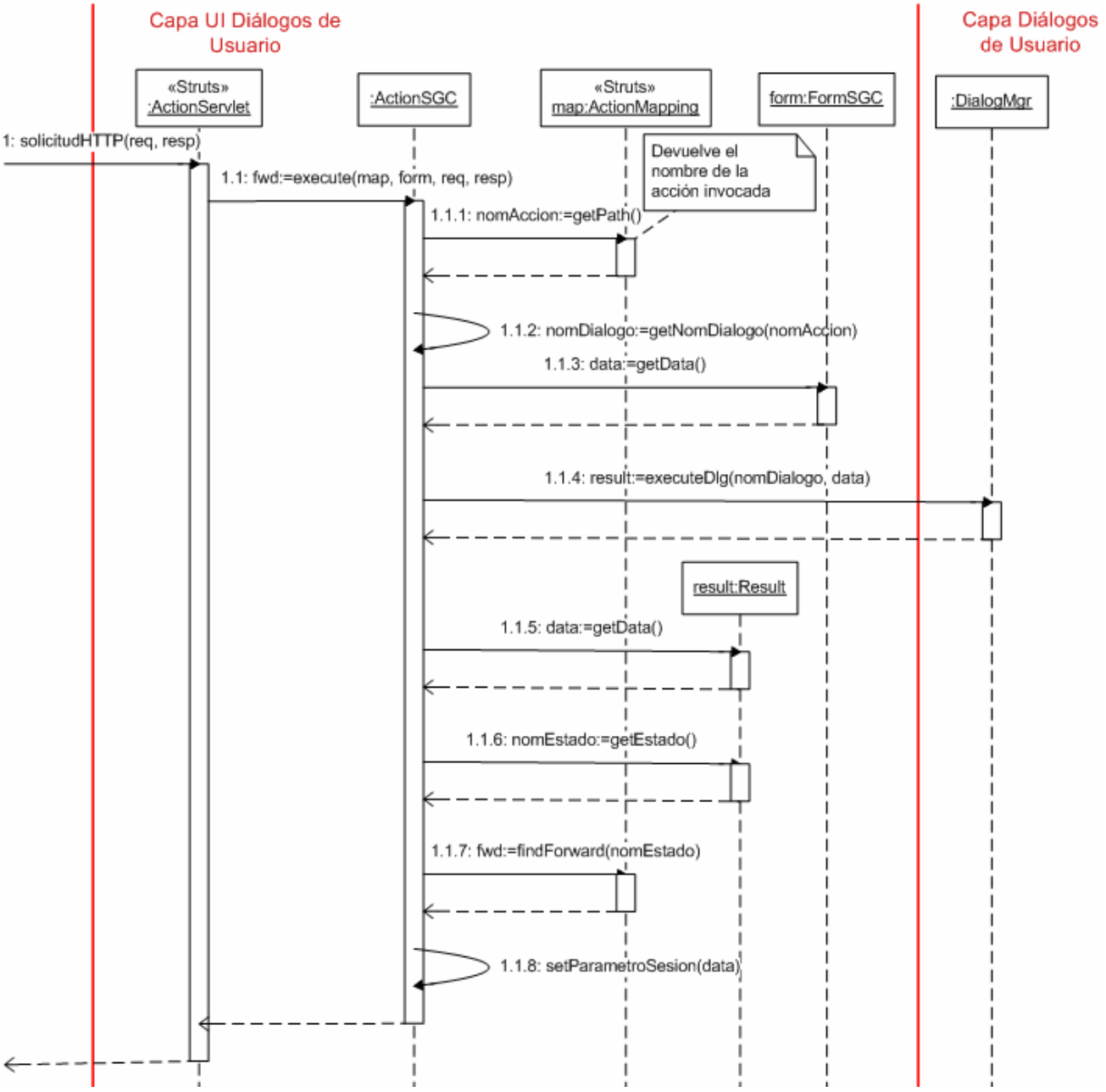


Figura 21 - Diagrama de secuencia para la capa UI Diálogos de Usuario

Recordemos que el control de las solicitudes es realizado por el ActionServlet proporcionado por Struts, el cual direccionará (según se ha configurado) cualquier solicitud a un objeto de la clase ActionSGC.

Al recibir una solicitud, el ActionServlet identifica el nombre de la acción enviada como parámetro por dicha solicitud. Luego crea un objeto ActionMapping con la información de configuración para la acción recibida, y un objeto ActionForm, en el que encapsula los datos enviados por la solicitud. La clase utilizada para el objeto ActionForm será la que se haya indicado en la configuración de Struts para esa acción.

Los objetos ActionMapping y ActionForm se pasan como parámetros al método execute() del objeto ActionSGC, junto con los objetos HttpRequest y HttpResponse recibidos por el Servlet.

El objeto ActionSGC obtiene el nombre de la acción invocada a partir del objeto ActionMapping. Luego obtiene el nombre del diálogo que debe ejecutar. Esto es posible gracias a la forma en que se ha definido la asignación de nombres para las acciones: *<nombre de caso de uso> / <nombre de acción>*

Dado que la metodología empleada propone el uso de objetos Data para la comunicación entre esta capa y la inferior, se optó por no utilizar los objetos ActionForm de Struts para este fin. Por lo tanto, los mismos son implementados de manera que encapsulen un objeto Data con los datos de cada página. El método getData de la clase ActionForm devuelve el objeto Data correspondiente a los datos que se estén manejando.

Con el nombre de diálogo y el objeto Data obtenidos, el ActionSGC ejecuta el diálogo correspondiente de la capa inferior.

A diferencia de lo que propone la metodología CBD, esta capa no se comunica en forma directa con cada diálogo, sino que lo hace a través de un nuevo componente denominado Dialog Manager. El funcionamiento de este componente se explicará al tratar la capa Diálogos de Usuario.

El resultado de la ejecución es un objeto de tipo Result tal como lo especifica la metodología. A partir de este objeto, el ActionSGC obtiene el estado en el que quedó el diálogo y el objeto Data que debe utilizar para generar la respuesta.

Dado que Struts utiliza nombres lógicos para referenciar las páginas, y que cada página representa un estado de un diálogo, se utilizan los nombres de los estados para definir dichas referencias. En términos de Struts, se define un ActionForward por cada estado. El método findForward de la clase ActionMapping devuelve un objeto ActionForward a partir de su nombre.

Finalmente, el ActionSGC coloca como parámetro de sesión el objeto Data devuelto por el diálogo ejecutado, y devuelve el ActionForward correspondiente a la siguiente página a mostrar.

Recordemos que el ActionServlet será el encargado de devolver la respuesta al usuario.

5.2.2. Capa Diálogos de Usuario

Los componentes de esta capa son los encargados de manejar la lógica de los casos de uso del sistema. Por cada caso de uso existe un componente, denominado Diálogo de Usuario, que conoce su máquina de estados y se encarga de llevar adelante su ejecución. Los diálogos son implementados como Stateful SessionBeans.

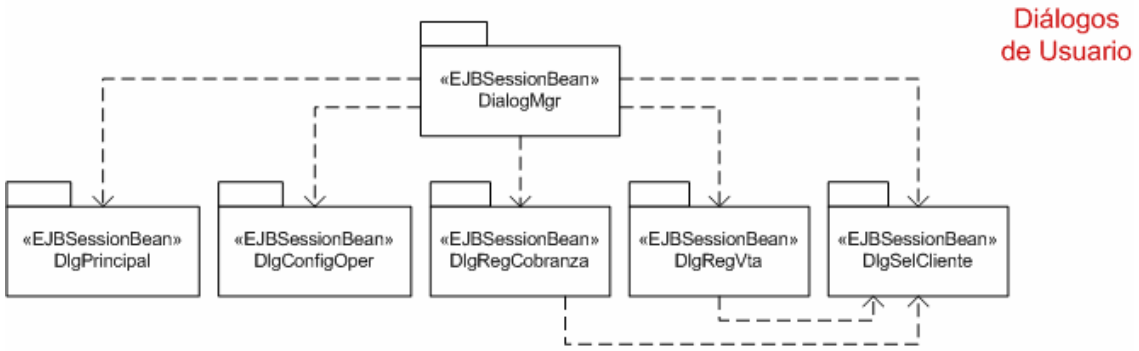


Figura 22 - Capa Diálogos de Usuario

Uno de los problemas que se intentó resolver en esta capa fue el de la inclusión de casos de uso, que en términos de implementación equivale a la composición de los diálogos correspondientes a esos casos de uso.

Este problema había sido tratado en [BFF03] pero la solución encontrada no es del todo flexible. La misma requería, en caso de que un diálogo tuviera que incluir a otro, que la clase Data usada por el diálogo contenedor implementara la interfaz de la clase Data usada por el diálogo incluido. Esto podía traer

inconvenientes si los objetos Data tenían propiedades con nombres iguales pero con significados semánticos diferentes. Dicha solución también podía resultar un tanto engorrosa en el caso en que se tuviese más de una inclusión anidada.

La solución que se plantea en este trabajo consiste en utilizar el componente Dialog Manager mencionado anteriormente, el cual se encarga de manejar las invocaciones entre los distintos diálogos cuando sea necesario. Este componente se implementa como un Stateful SessionBean.

Cada uno de los diálogos manejados por el Dialog Manager se corresponde con uno de los casos de uso comprendidos por el caso de estudio. Además se agrega un caso de uso que representa el estado del usuario en el sistema y que incluye a los casos anteriores. La máquina de estados correspondiente a este caso de uso es la siguiente:

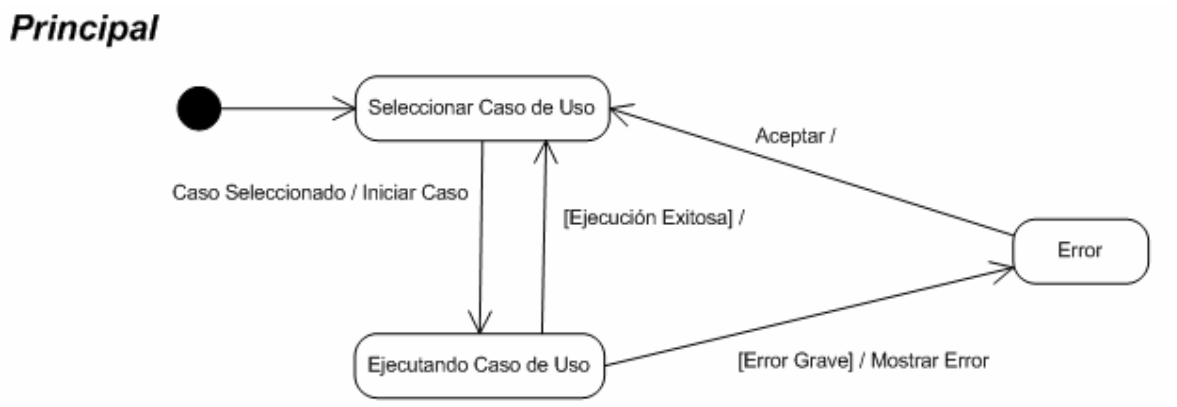


Figura 23 - Diagrama para el caso de uso Principal

Nota: *En los Anexos Particulares de este informe se podrán encontrar las máquinas de estados correspondientes al resto de los casos de uso.*

5.2.2.1. Clases de Datos

Se define una interfaz IData la cual es implementada por la clase Data, que es la clase base de los datos intercambiados entre los diálogos y los componentes de la capa superior.

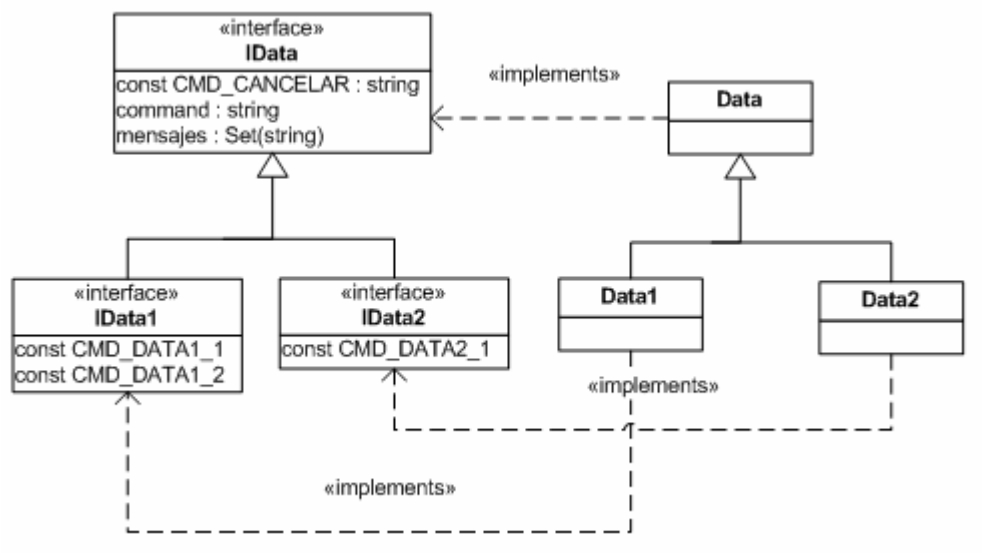


Figura 24 - Capa Diálogos de Usuario: Clases de Datos

Cada diálogo tendrá su propio conjunto de clases derivadas de la clase Data, cuyos nombres comienzan con el prefijo “Data”, con los datos necesarios para la ejecución de sus distintos estados. Al mismo tiempo cada clase DataX tendrá una interfaz IDataX (derivada de IData).

Una de las ideas adoptadas del trabajo presentado en [BFF03] fue la inclusión de la propiedad *command* en la clase IData, la cual indica la acción ejecutada por el usuario cuando se invoca el diálogo correspondiente. Esto es necesario ya que una misma pantalla puede tener que enviar su solicitud de ejecución al diálogo por distintos motivos.

Por ejemplo una página jsp que pertenece a un determinado caso de uso siempre invocará un mismo diálogo de usuario. Pero la página podría tener un botón "Aceptar" y otro "Cancelar", y ambos enviarían la solicitud de ejecución al diálogo al ser presionados. En este caso el atributo command del objeto Data enviado en dicha solicitud contendrá un valor que permitirá saber cual de los dos botones fue el que se presionó antes de ejecutar el diálogo.

En la interfaz IData se define el comando *Cancelar*, el cual es común a todas las interfaces que la extiendan, ya que casi cualquier pantalla presentará esta opción.

Las clases que extienden IData definen los comandos que se pueden invocar sobre el diálogo para ese conjunto de datos.

Otra propiedad que define la interfaz IData es *mensajes*. Esta propiedad es utilizada por los diálogos para devolver mensajes al usuario (dentro del objeto Data contenido en el objeto Result devuelto).

5.2.2.2. Clases de Diálogos

Cada diálogo es implementado como un Stateful SessionBean con las siguientes características:

- La clase Bean, que implementa la lógica del EJB, extiende a la clase Dialog. Dicha clase implementa funcionalidades comunes a utilizar por todos los diálogos.
- La interfaz del Bean extiende la interfaz IDialog propuesta por la metodología empleada.

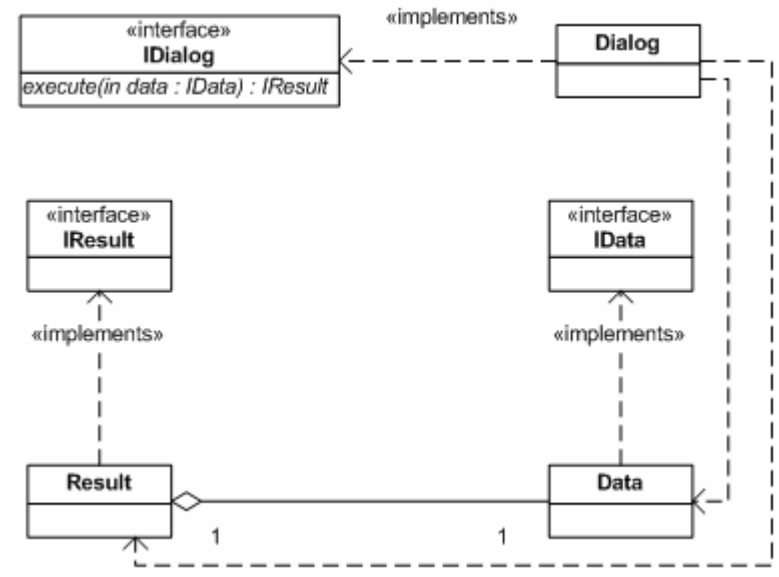


Figura 25 - Diagrama de clases base para la implementación de los Diálogos

No se empleó ningún patrón en particular para la implementación de los diálogos. Esto podría ser algo a tener en cuenta a la hora de mejorar este trabajo (por ejemplo se podía haber utilizado el patrón *State*, apto para implementar máquinas de estado).

Dos requerimientos importantes que tiene la manera elegida de implementar esta capa son que los diálogos tengan asociado un nombre único y que los estados de los diálogos también tengan un nombre único entre todos los diálogos.

Para cada diálogo se define una interfaz que comienza con el prefijo IDlg, en la que se definen las constantes necesarias para el funcionamiento del mismo. Como ejemplo veamos la interfaz del diálogo correspondiente a la selección de un cliente:

```
public interface IDlgSelCliente extends IDialog{
// Nombre del diálogo.
public static final String DLG_NAME           = "SelCliente";

// Estados del diálogo.
public static final String EST_SELECCION      = DLG_NAME + "/seleccion";
}
```

En esta interfaz se definen como constantes el nombre del diálogo y los nombres de los distintos estados del diálogo (en este ejemplo solo se define un estado). Los nombres de los estados de un diálogo se definen como el nombre del diálogo concatenado con un nombre que describa el estado, lo cual garantiza que el nombre del estado será único entre todos los diálogos, siempre que se cumpla que los nombres de diálogos son únicos.

La interfaz de un diálogo centraliza estos nombres para cualquier código que necesite referenciarlos, y se toma como referencia para los casos en los que no se usa código Java (por ejemplo los archivos de configuración que incluyan nombres de diálogos o de estados, deberán respetar los nombres definidos en estas interfaces).

En la interfaz IDialog (que es extendida por las interfaces IDlg particulares de los diálogos), se define un estado común a todos los diálogos que es el estado inicial. Este estado está pensado para uso interno de los diálogos y se asume que no tiene representación gráfica para el usuario, lo que equivale a decir que en la capa UI Diálogos de Usuario no existe ningún mapeo para dicho estado con ninguna vista.

5.2.2.3. Detalles del componente Dialog Manager

El Dialog Manager cumple con las siguientes funcionalidades:

- Implementa la única interfaz de comunicación con la capa superior UI Diálogos de Usuario. Los componentes de dicha capa interactúan con una instancia del componente Dialog Manager.
- Implementa el mapeo entre los nombres de los diálogos y los diálogos, para lo cual trabaja en conjunto con un archivo de propiedades en el que se define este mapeo.
- Mantiene las referencias a los diálogos en uso por parte de un determinado cliente.
- Maneja el stack de invocaciones en los casos en que un diálogo incluye a otro.

La interfaz provista por este componente es la siguiente:

IResult executeDialog (dialogName: String, data: IData)

A partir del nombre de diálogo el Dialog Manager puede obtener una referencia a dicho diálogo utilizando para ello el mapeo definido en el archivo de propiedades *Dialogs.properties*. Dicho archivo contiene para cada diálogo dos propiedades, la primera contiene el nombre JNDI del objeto Home correspondiente al diálogo y la segunda el nombre completamente calificado de la clase Home correspondiente al diálogo.

A modo de ejemplo se muestra un fragmento de este archivo:

```
##########   CONFIGURACION DE DIALOGOS  ##########

#
#  Para cada diálogo deben configurarse las siguientes propiedades:
#
#    dialogs.<nombre_dialogo>.homejndi=<nombre jndi de la home del dialogo>
#    dialogs.<nombre_dialogo>.homeclass=<ruta de la clase home del dialogo>
#
#  Ejemplo:
```

```
#
#     dialogs.RegVta.homejndi=java:comp/env/ejb/dialog/RegVta
#     dialogs.RegVta.homeclass=uy.com.isaltda.sgc.dialog.regvta.DlgRegVtaLocalHome
#

#####################################################
```

Cuando un cliente (desde la capa UI Diálogos de Usuario) se encuentra ejecutando un determinado caso de uso, todas las invocaciones de ejecución deben ser dirigidas a la misma instancia del diálogo que implementa ese caso de uso, ya que de otra forma se perdería el estado entre invocación e invocación.

Por este motivo, cada cliente tiene asignada una instancia del Dialog Manager, el cual mantiene las referencias al (o los) diálogo(s) en uso por parte de dicho cliente. Para ello es que se implementa dicho componente como un Stateful SessionBean.

Cuando un cliente invoca una ejecución sobre un determinado diálogo, la instancia del Dialog Manager que lo atiende controla si existe una referencia almacenada para ese diálogo, en cuyo caso utiliza dicha instancia. Si no existía una referencia almacenada se crea una nueva instancia para ese diálogo y se almacena.
El cliente sólo debe mantener una referencia a la instancia del Dialog Manager.

5.2.2.4. Invocación entre diálogos

Otra de las funcionalidades del Dialog Manager es la de manejar el stack de invocaciones entre diálogos cuando un diálogo necesita incluir a otro.

Para lograr la invocación entre diálogos se utilizan dos atributos en la clase Result: *infoEjecucion* y *subDialogo*.

El atributo infoEjecucion proporciona información acerca del estado resultante de la ejecución realizada, y podrá tener uno de los siguientes valores (definidos como constantes en la interfaz IResult) :

- **Ejecutando:** Indica que la ejecución del caso de uso no ha finalizado. El diálogo se encuentra en un estado intermedio y espera una nueva ejecución.
- **Invocación a diálogo:** Indica que el diálogo actual invocó a otro diálogo como resultado de la última ejecución.
- **Finalizado:** Indica que el diálogo ha finalizado su ejecución.

El atributo subDialogo se utiliza en conjunto con la segunda de las opciones anteriores, e indica el nombre del diálogo invocado por el diálogo actual.

El seudocódigo que define el comportamiento del Dialog Manager es el siguiente:

```
IResult  ejecutar_dialogo (nomDialogo, datos) {
      dialogo = buscar(nomDialogo)
      r = dialogo.execute (datos)

            if (r.infoEjecucion == INV_DIALOGO) {
            agrego_stack (nomDialogo)

            // Ejecutar el dialogo invocado con los datos del resultado
            return ejecutar_dialogo (r.subDialogo, r.data)

            }
            else if (r.infoEjecucion == FINALIZADO) {
                        if (! stack_vacio()) {
                              quito_stack(nomDialogoStack)

                              return ejecutar_dialogo (nomDialogoStack, r.data)
                        }
```

```
            }

      return r;
}
```

Como puede observarse este método presenta un comportamiento recursivo que refleja la utilización del stack tanto al realizar una llamada a un diálogo como al finalizar las mismas.

5.2.2.5. Detalles de distribución

Con respecto a la distribución de los componentes se asumió que los diálogos y los SessionBean de las capas inferiores se ubican en un mismo nodo, razón por la cual sólo dispondrán de interfaces locales. El componente Dialog Manager también se despliega en el mismo nodo, pero ofrece una interfaz remota, lo cual permite que se lo invoque desde un nodo diferente.

Esto alivia las transferencias de datos entre el Dialog Manager y los diálogos, y entre los diálogos y los servicios de capas inferiores, ya que si todos estos componentes tuvieran interfaces remotas, más allá de estar en un mismo servidor o no, el overhead producido por la serialización de los datos transferidos entre éstos afectaría negativamente el rendimiento del sistema en general.

El siguiente diagrama muestra el esquema básico:

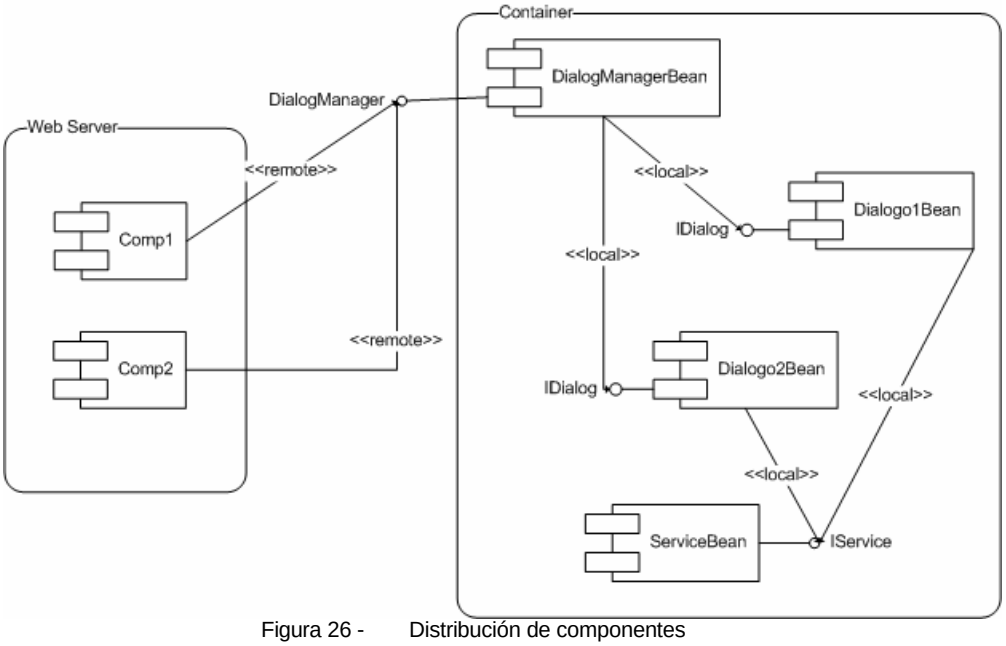


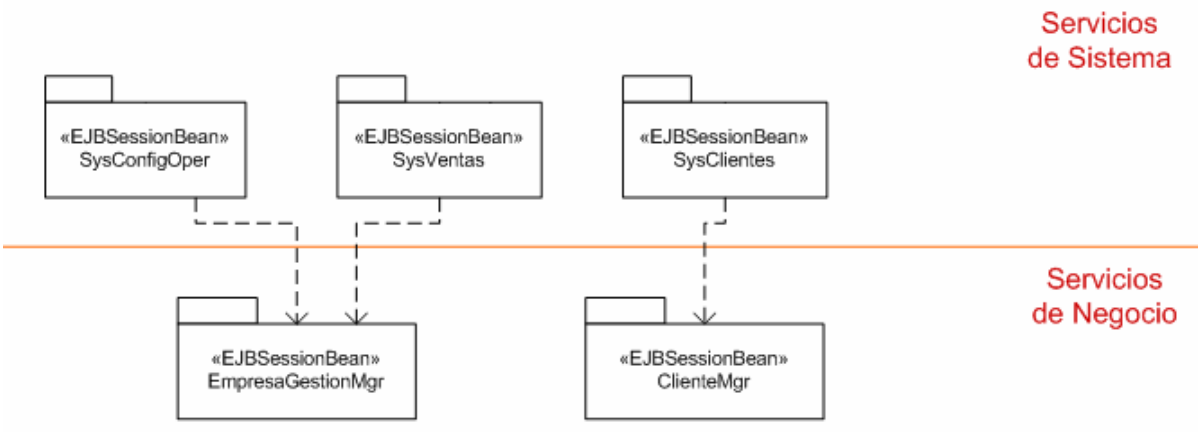
Figura 26 - Distribución de componentes

5.2.3. Capas de Servicios de Sistema y Servicios de Negocio

Estas capas se analizan de manera conjunta dado que las decisiones de implementación más importantes se centraron en las capas superiores.

Los componentes de la capa de Servicios de Sistema se encargan de agrupar las funcionalidades requeridas por los casos de uso correspondientes a un mismo proceso de negocio. Los mismos se implementan como Stateless SessionBeans.

En esta capa también se ubicarían los componentes encargados de manejar interfaces de sistemas externos. Como en el presente caso de estudio no se interactúa con ningún sistema, los componentes de dicha capa simplemente invocan los servicios proporcionados por la capa inferior.



Los componentes de la capa de Servicios de Negocio son los encargados de administrar los tipos estables del negocio (core types). Los core types detectados para el caso de estudio en cuestión son EmpresaGestion, que representa la empresa gestionada por el sistema, y Cliente, que representa una empresa cliente de la misma.

Para el manejo de los objetos persistentes se utilizó la alternativa de acceso a datos mediante JDBC utilizando el patrón DAO, el cual puede ser consultado en los anexos teóricos de este informe. Esto se debe a que para la mayor parte de dichos objetos no tiene mucho sentido crear un EJB de entidad, ya que sus datos corresponden a las tablas maestras del sistema y luego de cargarlos una primera vez serán accedidos a los efectos de consulta y escasas veces para su actualización.

5.3. Evaluación

La solución encontrada para el tratamiento de la inclusión de casos de uso ha provocado un pequeño cambio sobre las ideas de comunicación entre las capas UI Diálogos de Usuario y Diálogos de Usuario propuestas en [PRV03]. En dicho reporte se plantea la comunicación directa del componente de la capa UI con el diálogo que corresponda al caso de uso que se esté ejecutando. En nuestro caso, agregamos un nuevo componente dentro de la capa Diálogos de Usuario, el Dialog Manager, el cual centraliza la comunicación con la capa superior. Este nuevo componente no cambia la esencia de la arquitectura planteada.

Por otro lado, las adaptaciones realizadas a la implementación de Struts para lograr su alineación con la metodología CBD, provocan un mayor esfuerzo de implementación pero se logra mantener las ventajas de la utilización del framework, sin interferir con la aplicación de dicha metodología.

Con respecto a la capa de Servicios de Negocio, se deja planteada la necesidad de realizar un análisis más profundo de la alternativa seleccionada para su implementación.

Un aspecto importante que no fue mencionado en este capítulo es el manejo de transacciones en la capa de Diálogos de Usuario. La metodología CBD propone enmarcar la ejecución completa de un caso de uso dentro de una transacción. En los trabajos tomados como referencia este tema no parecía presentar muchos inconvenientes y debido a que se puso mayor énfasis en otros aspectos no fue tratado en nuestro trabajo. De todas formas este tema deberá ser considerado para un desarrollo completo del sistema.

----- ¤ -----

Capítulo 6. Caso Práctico

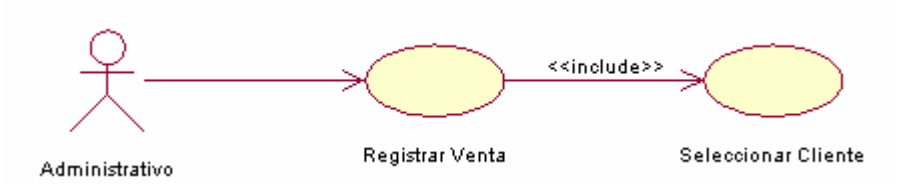
En este capítulo se describen los resultados obtenidos durante las etapas de análisis, diseño e implementación para uno de los casos de uso perteneciente al caso de estudio con el que se trabajó en forma práctica.

En los Anexos Particulares de este informe podrá encontrarse información más detallada correspondiente al caso de uso tratado aquí y al resto de los casos contenidos en el caso de estudio.

6.1. Caso de uso elegido

Uno de los casos de uso principales para el proceso de negocio involucrado en el sistema de gestión es el de poder registrar una venta realizada a un cliente.

Recordemos que este caso de uso incluye al caso de selección de un cliente, por lo cual también será tratado este caso.



6.1.1. Descripción expandida

A continuación se presentan las descripciones expandidas de ambos casos de uso, de acuerdo a la notación propuesta en [Lar99]:

Caso de Uso Nº 8	Registrar Venta
Actores	Administrativo
Propósito	Registrar una venta realizada a un cliente.
Resume n	El administrativo selecciona el cliente al que se realiza la venta, ingresa los datos de los productos vendidos e indica, entre otras cosas, si es una venta contado o crédito, la moneda y la forma de pago. El sistema calcula el total y registra la venta.
Tipo	Primario y esencial
Referencias cruzadas	Requerimientos: R 2.1, R 2.2, R 2.3, R 2.4, R 2.7, R 2.8, R 4.6, R 4.7 Casos de Uso: CU 8.1

Sección principal	
Curso normal de los eventos	
Acción del actor	**Respuesta del sistema**
1.Este caso comienza cuando un administrativo desea registrar una venta realizada a un cliente	2.El sistema presenta la pantalla con los datos requeridos para realizar dicho registro
3.El administrativo selecciona el cliente al que se le realiza la venta (Ver Caso de Uso 8.1)	
4.El administrativo selecciona la moneda en la cual se realiza la venta	5. El sistema habilita el ingreso de las líneas de venta
6.Para cada producto vendido el administrativo indica el ingreso de una línea de venta	7.El sistema muestra una lista con los productos para que se seleccione uno
8.El administrativo selecciona el producto deseado e indica la cantidad	9.El sistema carga el código, descripción y precio del producto seleccionado, permitiendo modificar el precio
10.El administrativo finaliza el ingreso de los datos	11.El sistema realiza el cálculo del importe total

	de la venta y presenta la lista de tipos de operaciones previamente configuradas para ventas
12.El administrativo selecciona el tipo de operación correspondiente.	13.El sistema asigna un número a la venta y la registra.

Caso de Uso Nº 8.1	Seleccionar Cliente
Actores	Administrativo
Propósito	Seleccionar el cliente con el cual se está realizando una operación
Resumen	El administrativo selecciona el cliente vinculado a la operación comercial que realizará en el sistema
Tipo	Primario y esencial

Sección principal	
Curso normal de los eventos	
Acción del actor	**Respuesta del sistema**
1.Este caso comienza cuando un administrativo desea seleccionar el cliente con el cual realiza una operación comercial	2.El sistema le presenta la lista de clientes registrados en él
3. El administrativo selecciona el nombre del cliente que corresponda	4.El sistema devuelve los datos relacionados con dicho cliente necesarios para la facturación

Nota: los requerimientos referenciados anteriormente en cada caso de uso, pueden encontrarse en el documento de especificación de requerimientos anexo a este informe.

6.1.2. Máquinas de estados

A raíz de las descripciones de los casos de uso realizadas anteriormente se pueden determinar una serie de estados por los que pasa el sistema durante su ejecución:

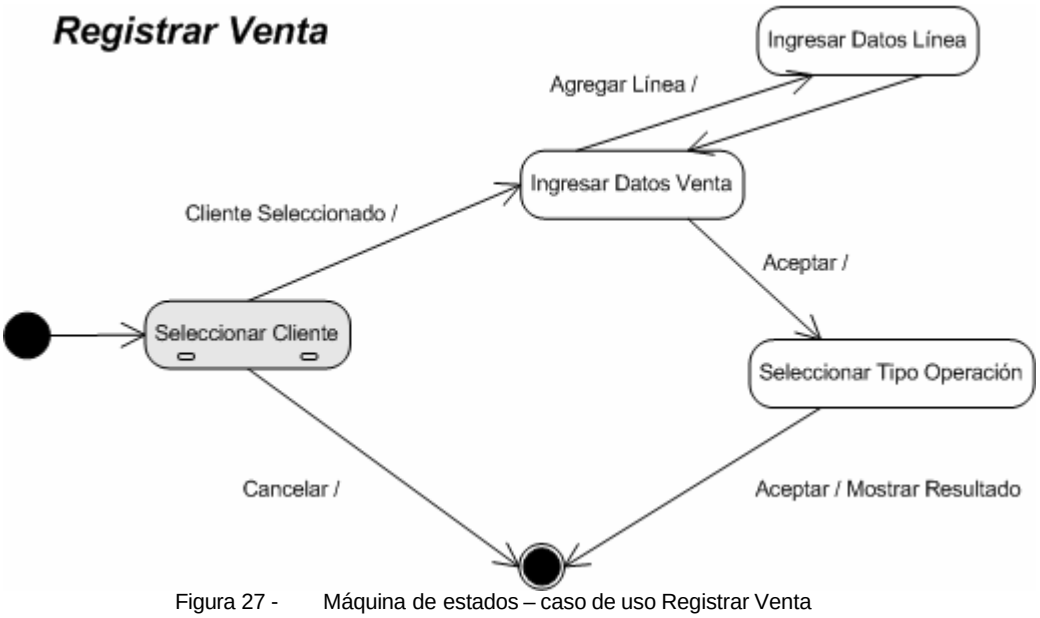


Figura 27 - Máquina de estados – caso de uso Registrar Venta

Al iniciar el caso de registro de venta, la primer tarea que debe realizarse es la de selección del cliente. Una vez seleccionado el mismo se puede comenzar con el registro de los datos asociados a la venta, en el que deberá ingresarse una línea por producto vendido. Por último deberá indicarse el tipo de operación comercial que se desea realizar, de acuerdo a las operaciones definidas en el sistema (venta contado, venta crédito, etc).

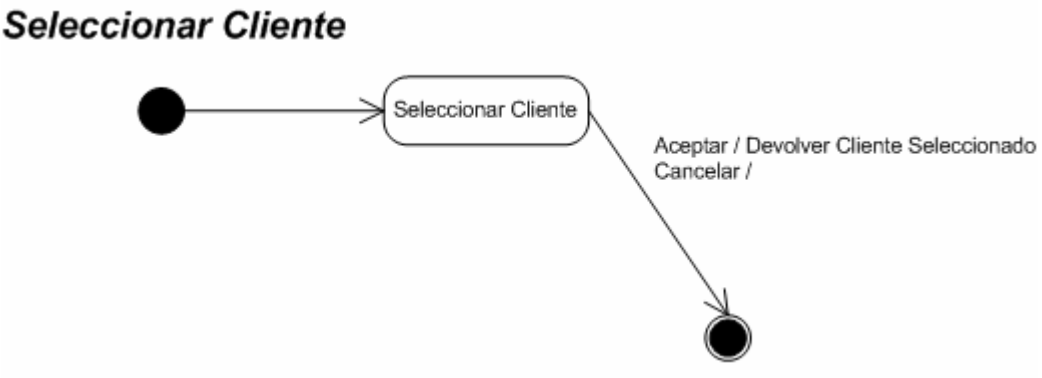


Figura 28 - Máquina de estados – caso de uso Seleccionar Cliente

La tarea de selección del cliente implica la presentación de la lista de clientes por parte del sistema; el usuario selecciona el que desea y el sistema carga los datos que correspondan a dicho cliente. A nivel de estados del sistema, solo existe uno que es el de seleccionar cliente.

6.2. Diseño

6.2.1. Si se utilizara Struts en su forma estándar...

En esta sección se presentará una visión general de lo que sería el diseño de los casos de uso mencionados si se utilizara el framework Struts tal como lo propone su funcionamiento original. Esto tiene como objetivo la clarificación de las diferencias existentes entre dicha propuesta y el diseño planteado en este proyecto.

Los elementos necesarios para los casos de uso planteados serían los siguientes:

- Tres páginas JSP:
  - regVta.jsp: incluye los campos necesarios para el ingreso de los datos correspondientes a una venta (moneda, forma de cobro, líneas de venta) y los botones que permiten realizar la selección del cliente y el agregado de los productos vendidos.
  - selCliente.jsp: incluye la lista de clientes disponibles en el sistema y el botón que permite seleccionar uno de ellos.
  - agregarLinea.jsp: incluye la lista de productos disponibles en el sistema y el botón que permite seleccionar uno de ellos.
- Tres JavaBeans que extiendan la clase ActionForm: uno por cada página JSP, con propiedades para obtener y establecer los valores de sus campos y el método de validación que corresponda para controlar los datos introducidos.
- Tres clases de acción que extiendan la clase Action: una por cada página.

Además se necesita una JSP, un ActionForm y una clase Action para lo que sería la pantalla inicial del sistema, en la cual podrían seleccionarse las distintas opciones del mismo.

WebSphere permite dibujar un diagrama con los elementos de la aplicación Struts para luego ir generándolos en forma automática a partir de allí. Al utilizar Struts en su formato estándar, dicho diagrama se utiliza para diseñar el flujo que tiene la aplicación entre sus distintos componentes.

El flujo que correspondería a los casos de uso tratados en este capítulo sería el siguiente:

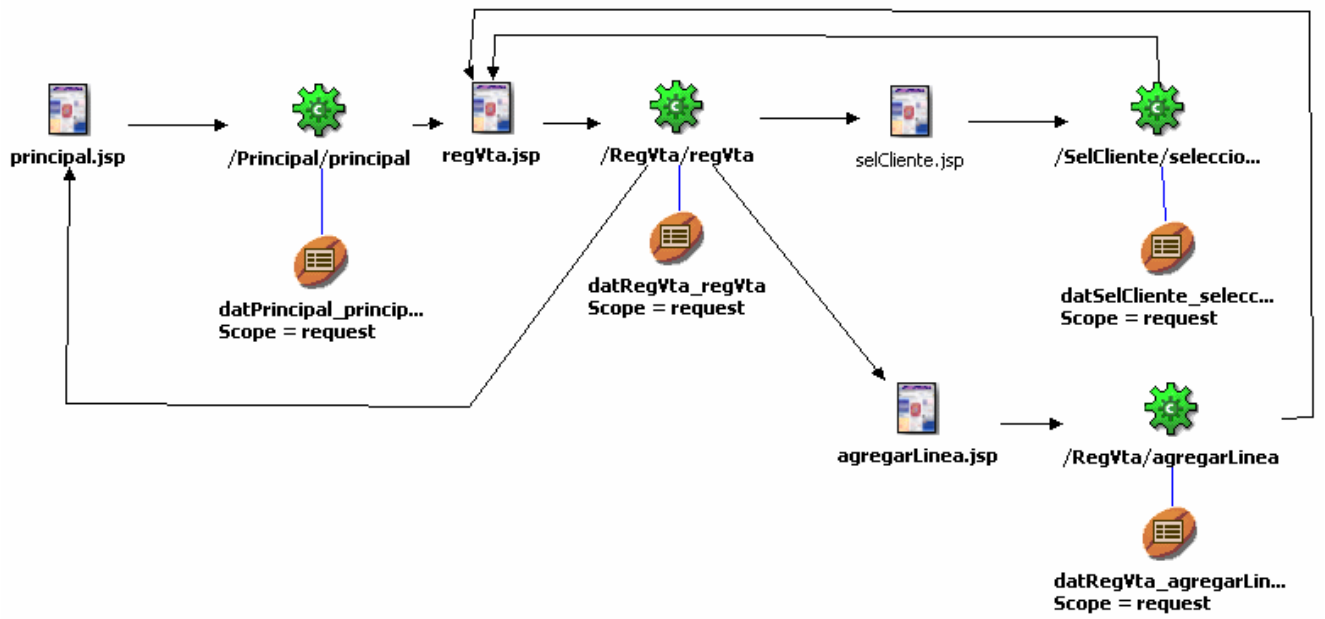


Figura 29 - Diagrama de interacción de componentes Struts

De esta forma se establece que desde la acción ejecutada en la página principal se abre la página regVta (si estuviéramos representando otros casos de uso habría otras opciones).

Desde la clase Action asociada a la página regVta la aplicación puede dirigirse a la página selCliente (si se presiona el botón de seleccionar cliente), a la página agregarLinea (si se presiona el botón de agregar línea) o nuevamente a la página principal (si se finaliza el registro de la venta). En este diagrama se establecen todos los flujos posibles; luego en el código se indicará cuándo se realiza el forward a cada una de las páginas.

Se puede observar que las acciones asociadas a las páginas selCliente y agregarLinea redirigen el flujo únicamente hacia la página de registro de venta y solo son llamadas por las acciones de dicha página. Esto es porque la página agregarLinea forma parte del caso de uso Registrar Venta y la página selCliente pertenece al caso Seleccionar Cliente, el cual está incluido en el de Registrar Venta.

6.2.2. Siguiendo la metodología de desarrollo basado en componentes

Al utilizar la implementación propuesta en este proyecto se tomó la decisión de que las clases Action no se encarguen de invocar las operaciones correspondientes a la ejecución del caso de uso, sino que ejecuten el diálogo de usuario que corresponda, y que sea éste el responsable de mantener el estado y solicitar los servicios necesarios a las capas inferiores.

Esto llevó a implementar una única clase Action que determina cuál es el diálogo que debe ejecutar en cada caso. El resto de los componentes Struts necesarios para esta modalidad, son los mismos que los mencionados en el caso anterior (páginas JSP y ActionForms). El flujo de interacción entre esos componentes ya no se diseñará en el diagrama, sino que será manejado por la capa inferior.

En este caso, el diagrama solo representará la relación entre los componentes propios de Struts como se muestra a continuación:

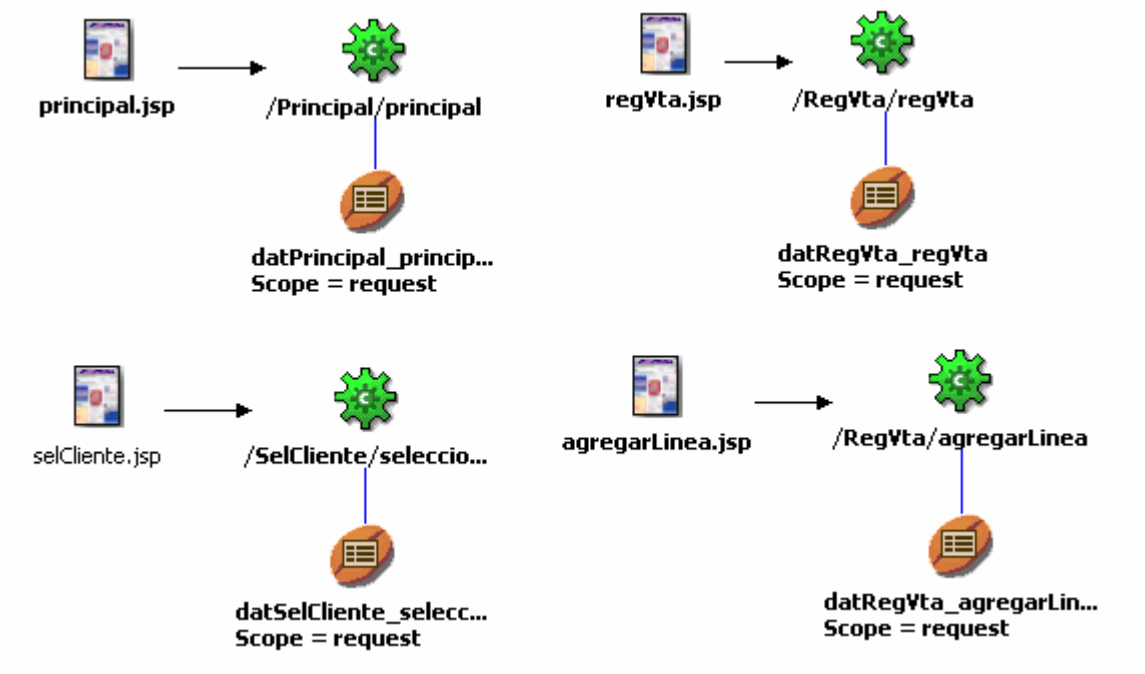


Figura 30 - Diagrama de componentes Struts utilizando la metodología CBD

Las JSPs pertenecen a la capa superior de la arquitectura (Interfaz de Usuario), mientras que los ActionForms y las clases Action se ubican en la capa UI Diálogos de Usuario.

La arquitectura para el resto de las capas es la siguiente:

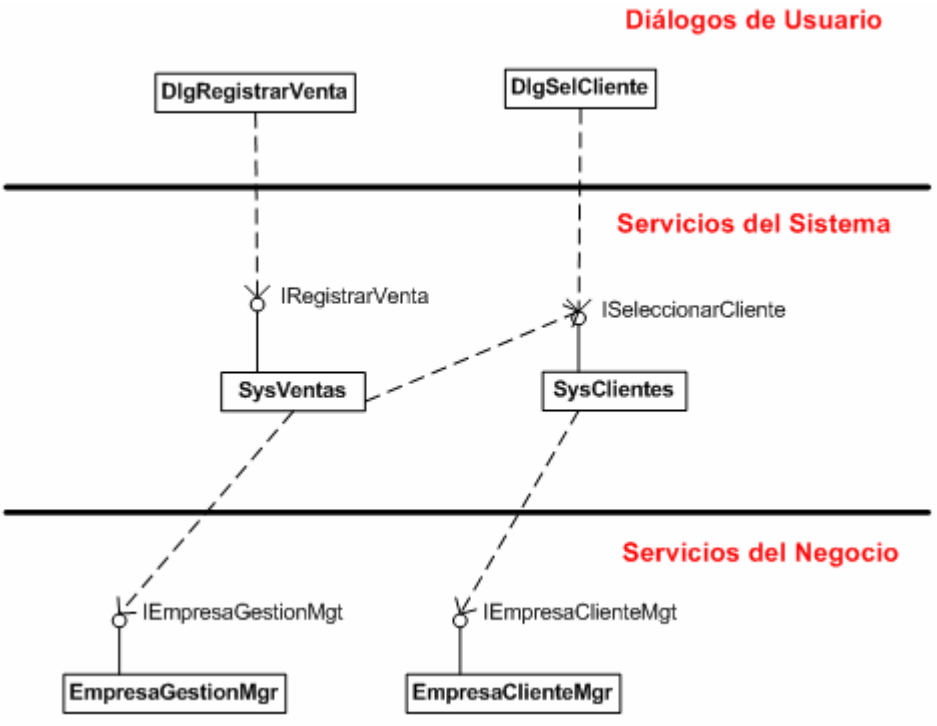


Figura 31 - Arquitectura para el caso de ejemplo

La capa de Diálogos de Usuario contiene un Stateful SessionBean por cada caso de uso (DlgRegVta y DlgSelCliente), el cual se encarga de manejar la máquina de estados del caso de uso, y uno que se encargue de manejar la lógica de la pantalla principal (DlgPrincipal).

La capa de Servicios de Sistema contiene dos Stateless SessionBeans, uno correspondiente al componente asociado con el proceso de ventas (SysVentas) y otro que corresponde al componente relacionado con la manipulación de clientes (SysClientes). Estos componentes deben ofrecer los servicios requeridos por la capa superior para llevar a cabo la ejecución de los distintos casos de uso. Las interfaces detectadas para cada uno de ellos son las siguientes:

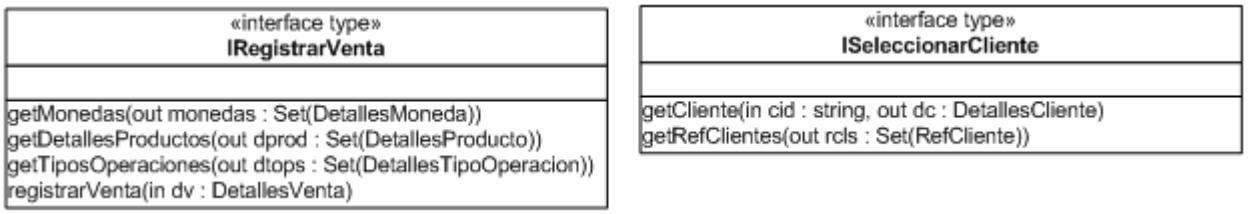


Figura 32 - Interfaces de los servicios del sistema.

Las operaciones ofrecidas por estas interfaces no se implementan en la capa Servicios de Sistema, sino que son provistas por la capa Servicios de Negocio.

En la capa de Servicios de Negocio se ubican los Stateless SessionBeans encargados de administrar las operaciones básicas del sistema y aspectos de persistencia: EmpresaGestionMgr, que maneja los datos relacionados con la empresa que gestiona el sistema, y ClienteMgr, que se encarga de manipular los datos de los clientes.

6.3. Evaluación

En este capítulo se han llevado a la práctica las ideas teóricas presentadas en los capítulos anteriores, de forma tal que se puedan visualizar de una forma más clara los componentes concretos para un caso de uso.

----- ¤ -----

Capítulo 7. Conclusiones y Trabajo Futuro

En este capítulo se realiza un análisis del cumplimiento de los objetivos del proyecto y se plantean dificultades y soluciones encontradas en el transcurso del mismo que llevaron a modificar el lineamiento inicial.

Además se detallan los aportes del proyecto a nuestra formación profesional y el trabajo que se realizará en un futuro para ampliar las metas alcanzadas.

Por último, se presentan una serie de pautas que pueden guiar el desarrollo de aplicaciones J2EE en la empresa.

7.1. Trabajo realizado y cumplimiento de objetivos

El trabajo realizado durante este proyecto se puede resumir en los puntos que se detallan a continuación:

7.1.1. Investigación J2EE

El objetivo principal de investigar el estándar J2EE y la conveniencia de la utilización de cada uno de sus conceptos fue cumplido.

Además, se analizaron las ventajas de la utilización de frameworks en la capa web de la aplicación y se adquirió el conocimiento del funcionamiento de Struts.

Resultados tangibles de la investigación realizada son los documentos, anexos teóricos de este informe, que describen:

- la arquitectura J2EE y los patrones de diseño que involucra
- los distintos conceptos J2EE y su utilización
- ventajas de utilización de frameworks
- funcionamiento de Struts

7.1.2. Análisis del Sistema de Gestión

Se cumplió con el objetivo de analizar los requerimientos del nuevo sistema de gestión con interfaz web para la empresa.

La realización del análisis completo del nuevo sistema ha dejado como resultados tangibles los documentos de especificación de requerimientos, casos de uso y modelo conceptual, que se presentan en los anexos particulares de este informe.

Esta tarea permitió contar con una visión global del sistema, la cual se utilizó para crear un diseño que contemplara las necesidades de todo el sistema.

7.1.3. Diseño e Implementación del Sistema de Gestión

Este objetivo se cumplió en forma parcial, ya que no se ha realizado el diseño y la implementación del sistema completo de gestión, sino que se han abordado un grupo de casos de uso del mismo para construir un prototipo. Las razones de ello se presentan en la próxima sección de este capítulo.

A raíz de la investigación de la metodología de desarrollo basado en componentes [VP03] y las alternativas de diseño y/o implementación para alinear J2EE y Struts a ella, se lograron establecer pautas para el diseño de aplicaciones J2EE en la empresa.

Un detalle importante es que dichas pautas se centran en la realización de un diseño con el menor nivel de acoplamiento posible entre las distintas capas de la arquitectura. Se hace uso de las ventajas proporcionadas por el framework pero sin ligar la aplicación a su utilización.

El prototipo construido proporciona una implementación de la arquitectura que consideramos deseable para la construcción de futuras aplicaciones, permitiendo además la reutilización de sus componentes básicos.

Otro objetivo cumplido fue el de interiorizarnos con el ambiente de desarrollo WebSphere Studio Application Developer, el cual será incorporado próximamente en la empresa.

Además del prototipo desarrollado, otro resultado tangible es el documento anexo que describe detalles de la implementación J2EE en WebSphere, información que permitirá desarrollos futuros más rápidos.

7.2. Cambios en el lineamiento inicial

La presentación de este proyecto surgió por la necesidad de incorporar la tecnología J2EE a la empresa ISA Ltda. En busca de un caso práctico que facilitara la compresión de dicha tecnología, y dada la necesidad de la empresa de contar con un sistema que permita realizar su gestión en forma centralizada, se propuso además el análisis, diseño e implementación del sistema de gestión web.

Al inicio del proyecto se comenzó con el análisis del sistema y la investigación de J2EE en forma paralela. Luego de realizar dicho análisis y de tener un primer acercamiento a la tecnología J2EE, se concluyó que la carga de trabajo requerida por las actividades propuestas inicialmente era demasiada. Por un lado, la cantidad de requerimientos analizados fue mayor a la que se había previsto. Por el otro, al investigar la tecnología J2EE, se encontró un tanto compleja la iniciación en ese tema debido a que fue necesario asimilar una gran variedad de conceptos de manera previa a su utilización.

Esto llevó a que se replantearan los objetivos del proyecto, de modo de poder profundizar en aspectos teóricos relacionados con el estándar J2EE y las herramientas de desarrollo, acotando el desarrollo del sistema de gestión a la realización de un caso de estudio. Es así como se decidió trabajar en los siguientes aspectos:

- Investigación de la tecnología J2EE.
- Búsqueda de metodologías aplicables a la tecnología.
- Utilización de herramientas para el desarrollo de aplicaciones J2EE.

Como el objetivo fundamental era el de lograr que la empresa pudiera iniciarse en la utilización de J2EE a partir de este trabajo, se decidió dedicar tiempo exclusivo a investigar dicho estándar, e incluso analizar las mejores prácticas de su utilización.

Luego de finalizada la investigación se vio la necesidad de buscar una metodología que se adaptara a la arquitectura J2EE. Se consideró interesante la posibilidad de aplicar la metodología de desarrollo basado en componentes propuesta en [VP03], para la cual se disponía de otro trabajo centrado en los componentes de J2EE [PRV03].

Adicionalmente se investigó la posibilidad de adoptar el framework Struts, para lo cual se debió analizar la manera más conveniente de alinear la utilización del mismo conjuntamente con la metodología mencionada en el párrafo anterior.

Esto provocó un cambio de versión de la herramienta de desarrollo, ya que la versión de WebSphere utilizada en la etapa de investigación para realizar las pruebas (versión 4) no brindaba soporte para la creación de aplicaciones en base a dicho framework, como es el caso de la versión 5.

Por este motivo, en la etapa de implementación nos enfrentamos con el hecho de que varias de las interfaces para la creación de los objetos y para la realización de configuraciones eran diferentes respecto a las que se habían estudiado inicialmente. A su vez, durante la implementación tuvimos que tomar una serie de decisiones sobre la forma de atacar determinadas situaciones que podrían darse en el funcionamiento de cada capa de la arquitectura y en la comunicación entre las distintas capas. Todo esto provocó que la implementación también llevara más tiempo del previsto.

7.3. Aportes del proyecto

Durante el desarrollo de este proyecto de grado hemos transitado por todas las etapas que implica la realización de un proyecto y asumido los diferentes roles que se necesitan para lograr llevarlo adelante en forma exitosa.

Hemos pasado por las reuniones con los integrantes de la empresa para realizar el relevamiento de requerimientos del nuevo sistema, la realización del análisis de dicho sistema, la planificación de las tareas del proyecto y la necesidad de enfrentar su modificación a medida que íbamos avanzando, la necesidad de acotar el caso práctico previsto a implementar inicialmente, la investigación y utilización de nuevas tecnologías, la realización del diseño del caso de estudio utilizando una nueva metodología de desarrollo y la necesidad de realizar adaptaciones a la forma de funcionamiento de un framework (también nuevo para nosotros) a la hora de la implementación del caso de estudio, la utilización de una nueva herramienta de desarrollo, etc.

Todo esto ha permitido que lográramos tener una visión de las dificultades que pueden llegar a tener cada una de las personas que participan en la realización de un proyecto, lo que consideramos aporta muchísimo valor a nuestra formación profesional. Sería muy difícil ser líder de un proyecto si solo se conocieran desde adentro algunos de los roles involucrados en el mismo.

A su vez, el hecho de que este proyecto se haya orientado al uso de metodologías de desarrollo, tecnologías y herramientas que hasta ahora desconocíamos nos ha aportado una gran cantidad de conocimientos muy valiosos para el futuro de nuestra profesión, así como también nos ha permitido entrenarnos en la incorporación de nuevos conceptos, tarea que debe realizar frecuentemente un profesional de nuestra área.

Desde el punto de vista de la empresa, creemos que este proyecto ha sido un gran avance para facilitar la futura utilización del estándar J2EE y del ambiente de desarrollo WebSphere. Hemos logrado crear documentos que los demás integrantes podrán utilizar para comenzar a introducirse en el mundo J2EE, tenemos pautas sobre las metodologías de desarrollo convenientes, conocemos las ventajas de la utilización de frameworks y el funcionamiento de uno de ellos, nos hemos enfrentado con varias dificultades al momento de programar con WebSphere y sabemos cómo solucionarlas, etc.

7.4. Trabajo futuro

Una de las características en que hicimos hincapié al diseñar la aplicación, como se ha mencionado anteriormente, fue en el desacoplamiento de las distintas capas que componen su arquitectura. La independencia lograda permite plantearnos como trabajo futuro el profundizar en la estructura de alguna de ellas que por razones de tiempo no pudimos profundizar lo suficiente. Por ejemplo, la capa de servicios de negocio ha quedado un poco relegada hacia el final y, a pesar de que analizamos las distintas alternativas para el manejo de la persistencia de los datos, al momento de implementar el caso de estudio no hemos tomado una decisión firme sobre cuál de ellas utilizar. Otro tema que se dejó pendiente fue el manejo de transacciones para la capa de diálogos de usuario, el cual deberá ser profundizado.

Luego de la finalización de este proyecto se mejorará la definición de la arquitectura para el desarrollo de una aplicación J2EE, intentando utilizar la mejor opción, o la más cercana a ella, en cada una de las capas.

A su vez, se continuará con el diseño e implementación de los casos de uso pendientes para terminar con la construcción del sistema de gestión Web para la empresa.

7.5. Recomendaciones para futuros desarrollos J2EE en la empresa

Luego de la investigación de J2EE y de las metodologías que podríamos aplicar para la realización del análisis y diseño de aplicaciones empresariales, podemos realizar las siguientes recomendaciones:

Se recomienda complementar la metodología de análisis y diseño planteada en [Lar99] con otros trabajos que agreguen conceptos no tratados por la misma, como por ejemplo el enfoque de modelado de negocio propuesto en [Mol00].

Otra alternativa, que puede adecuarse un poco más a la tecnología J2EE, es la propuesta realizada en [VP03], que proporciona un enfoque para el desarrollo basado en componentes de aplicaciones empresariales de gran porte.

En nuestro caso se aplicó una combinación de ambos enfoques, principalmente porque estas metodologías se fueron investigando a medida que se desarrolló el análisis del caso de estudio.

Con respecto al desarrollo en sí, luego de interiorizarnos con los distintos conceptos de J2EE y analizar las alternativas para lograr un mejor aprendizaje y utilización de los mismos, podemos decir que sería deseable que cuando se realicen aplicaciones J2EE se hagan sobre un framework preexistente que utilice los patrones de diseño J2EE, de forma tal que se puedan aprovechar sus beneficios.

WebSphere Studio Application Developer V5.0 o posterior, herramienta en particular que se utilizará en la empresa, soporta la creación de aplicaciones basadas en el framework “Apache Struts”.

Según [SSJ02], como Struts es altamente configurable y tiene una larga y creciente lista de características, puede considerarse como el framework de mayor firmeza en la industria y es apropiado para grandes aplicaciones.

Como el uso de Struts tiene cierto impacto sobre la arquitectura lógica de la aplicación, no es posible que se utilice la metodología propuesta en [VP03] y el mapeo realizado en [PRV03] en su forma original. Una opción posible de diseño e implementación es la utilizada en este proyecto, en la cual se obtienen las ventajas del framework al utilizarlo, pero a su vez se realiza su implementación de forma tal de que se mantenga la independencia entre las distintas capas de la arquitectura, permitiendo que en un futuro se puedan realizar modificaciones que no causen impactos sobre las demás capas de la aplicación.

----- ¤ -----

Referencias Bibliográficas

- **[Alt03]** – Struts, Implementación del patrón MVC en aplicaciones Web – Pello Altadill – 2003 - http://pello.info/?p=struts.html (último acceso: 01/08/03)
- **[Ant01]** – Manual Básico de Struts – Javier Antoniucci – 2001 - http://www.antoniucci.com.ar/javier/struts.html (último acceso: 23/02/04)
- **[Bel03] –** Material del Curso de Integración de Sistemas - Fernando Bellas – Facultad de Informática de A. Coruña - 2003 - http://www.tic.udc.es/~fbellas/teaching/is/ (último acceso: 16/01/04)
- **[BFF03]** – Implementación de un Sistema de Información basado en componentes con J2EE – Sebastián Bengoechea, Fabián Fajardo, Juan Ferré - Instituto de Computación, Facultad de Ingeniería, Uruguay – 2003
- **[Cav02]** – Jakarta Struts 1.1 – Ready for Prime Time – Chuk Cavaness – 2002 - http://jakarta.apache.org/struts/resources/articles.html (último acceso: 25/09/03)
- **[CD01]** – UML Components: A Simple Process for specifying Component-Based Software – J. Cheesman, J. Daniels – Addison Wesley - 2001
- **[Cer03]** - Casos de Uso - Un Método Práctico Para Explorar Requerimientos – Santiago Ceria – Universidad de Bs. As. - http://www.dc.uba.ar/people/materias/isoft1/apuntes/casosdeuso.pdf (último acceso: 22/04/03)
- **[Cet02]** - Taller de Introducción a las plataformas J2EE y .NET – www.cetenasa.es/e-business/Talleres/taller2/book1.html (último acceso: 15/04/03)
- **[Cru03]** – Core J2EE Patterns & Micro Architectures – John Crupi – 2003 – www.richmondjug.com/presentations/CoreJ2EEPatterns.pdf (último acceso: 05/09/03)
- **[Dav01]** – Struts: an open-source MVC implementation – Malcom Davis – 2001 – http://www-106.ibm.com/developerworks/ibm/library/j-struts/ (último acceso: 25/09/03)
- **[Eck00]** – Thinking in Java 2º Edition – Bruce Eckel - Prentice Hall - 2000
- **[Hol02]** – Struts: a Solid Web-App Framework – Tim Holloway – 2002 - http://www.fawcette.com/javapro/2002_04/magazine/features/tholloway/default_pf.aspx (último acceso: 18/11/03)
- **[Hus 02]** – Use EJBs with care – Struts Tips – 2002 - www.husted.com/struts/tips/018.html (último acceso: 16/01/04)
- **[J2EE]** – Java 2 Platform, Enterprise Edition – Home page - http://java.sun.com/j2ee/
- **[Jak03]** - Struts: How to Access a Database – Apache Software Foundation – 2003 - http://jakarta.apache.org/struts/faqs/database.html (último acceso: 19/01/04)
- **[JWord02]** - Rumble in the Jungle: J2EE versus .Net - Java World - 2002 - http://www.javaworld.com/javaworld/jw-06-2002/jw-0628-j2eevsnet-p2.html (último acceso: 24/08/03)
- **[Lar02] –** Transparencias del curso de "Taller de Sistemas de Información 2" – Gustavo Larriera – Facultad de Ingeniería, Uruguay – 2002 - http://www.fing.edu.uy/inco/cursos/tsi2/TSI2-Clase5dotnet.pdf (último acceso: 24/08/03)
- **[Lar99]** - UML y Patrones / Introducción al análisis y diseño orientados a objetos - Craig Larman – Prentice Hall, México – 1999
- **[Lop03]** – Análisis estructurado: Modelo Entidad-Relación – Juan López – Universidad de Murcia 2003 - http://dis.um.es/~lopezquesada/documentos/Tema%203%20parte%204.ppt (último acceso: 13/02/04)
- **[Mar02]** - EJB Design Patterns / Advanced Patterns, Processes and Idioms – Floyd Marinescu – John Wiley & Sons – 2002
- **[Mat02]** – Using Struts – Larry Maturo – Athens Group (http://athensgroup.com), Austin – 2002 - http://stealthis.athensgroup.com/presentations/Model_Layer_Framework/Struts_Whitepaper.pdf (último acceso: 23/02/04)

- **[Mol00]** – De los Procesos del Negocio a los Casos de Uso – Jesús Molina García – Facultad de Informática, Universidad de Murcia – 2000 - http://dis.um.es/~jmolina/jis2000modeladonegocio.pdf (último acceso: 23/02/04)
- **[Per02]** - Desarrollo de Aplicaciones Web con Struts – Daniel Pérez Berenguer – Facultad de Informática, Universidad de Murcia – 2002 - http://dis.um.es/~jmolina/pfc.html (último acceso: 23/02/04)
- **[Peral03]** – Transparencias del curso Taller de Sistemas de Información I: Arquitecturas Web – Verónika Peralta – Facultad de Ingeniería, Uruguay – http://www.fing.edu.uy/inco/cursos/tsi1/teorico/10-web.pdf (último acceso: 17/11/03)
- **[PRV03]** – Arquitectura de Sistemas de Información basados en Componentes sobre la Plataforma J2EE – Daniel Perovich, Leonardo Rodríguez, Andrés Vignaga – InCo, Facultad de Ingeniería, Uruguay – 2003
- **[PV03]** – SAD del Subsistema de Reservas del Sistema de Gestión Hotelera – Daniel Perovich, Andrés Vignaga – Reporte técnico RT-03015, InCo, Facultad de Ingeniería, Uruguay – 2003
- **[Ram02]** - Design Patterns for Building Flexible and Mantainable J2EE Applications – Vijay Ramachandran - 2002 - http://developer.java.sun.com/developer/technicalArticles/J2EE/despat/ (último acceso: 23/02/04)
- **[SSJ02]** - Designing Enterprise Applications with the J2EE Plataform, Second Edition - Inderjeet Singh, Beth Stearns, Mark Johnson – Addison Wesley - 2002 (a la fecha 02/10/03 se encontraba disponible en http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/DEA2eTOC.html)
- **[Sun]** – Sun Microsystems – Home page - http://www.sun.com
- **[Sun00]** – The Java 2 Enterprise Edition Developer Guide, Version 1.2.1 – 2000 – http://java.sun.com/j2ee/ (último acceso: 23/02/04)
- **[Struts]** – The Apache Struts Web Application Framework – Home page - http://jakarta.apache.org/struts/
- **[Uml]** – UML home page - www.uml.org
- **[VP02]** – Desarrollo Basado en Componentes – Andrés Vignaga, Daniel Perovich – Facultad de Ingeniería, Uruguay - 2002
- **[VP03]** - Enfoque Metodológico para el Desarrollo Basado en Componentes – Andrés Vignaga, Daniel Perovich – Reporte técnico RT-03-14, InCo, Facultad de Ingeniería, Uruguay – 2003
- **[WF]** – Plataforma de software WebSphere - http://www.ibm.com/uy/products/software/websphere/
- **[WF31102]** - Manuales del curso "Servlet and JSP Development for WebSphere using WebSphere Studio Application Developer V4.0.2" – IBM Learning Services - 2002
- **[WF3502]** - Manuales del curso "Enterprise JavaBean (EJB) Development using WebSphere Studio Application Developer V4.0" – IBM Learning Services – 2002
- **[WSAD02]** - WebSphere Studio Application Developer Programming Guide – IBM Redbook SG24-6585-00 - 2002
- **[WSAD03]** - WebSphere Studio Application Developer Version 5 Programming Guide – IBM Redbook SG24-6957-00 – 2003

----- ¤ -----

Glosario

❖ A

Actor
Entidad externa de un sistema que de alguna manera participa en la historia de un caso de uso. Puede ser una persona, otro sistema o algún aparato eléctrico o mecánico que interactúa con el sistema.

Análisis
Investigación de un dominio que da origen a modelos que describen sus características estáticas y dinámicas. Se centra en cuestiones de "qué" más que de "cómo".

Análisis orientado a objetos
Investigación de un dominio o sistema de problemas a partir de los conceptos de dominio, como tipo de objetos, asociaciones y cambio de estado.

Apache Struts
Es un marco de trabajo (framework) de código abierto para construir aplicaciones Web, que se basa en el paradigma de diseño MVC (Modelo/Vista/Controlador). Se puede encontrar mayor información en [Struts].

API
Application Programming Interface. Una serie de reglamentos y acuerdos que nos definen la forma de llamar determinado servicio desde cierto programa.

Aplicación
Cada uno de los programas que, una vez ejecutados, permiten trabajar en una computadora. Son aplicaciones los procesadores de textos, hojas de cálculo, bases de datos, programas de dibujo, paquetes estadísticos, etc.

Application Developer
Forma reducida de hacer referencia a la herramienta WebSphere Studio Application Developer.

Artefacto
Información que es utilizada o producida mediante un proceso de desarrollo de software. Pueden ser artefactos un modelo, una descripción o un software.
Los artefactos de UML se especifican en forma de diagramas; éstos, junto con la documentación sobre el sistema, constituyen los artefactos principales que el modelador puede observar.

❖ B

Base de datos
Es un repositorio de información organizada de tal forma que permite un rápido acceso y búsqueda de dicha información.

Base de datos relacional
Un método de organizar bases de datos que crea relaciones entre archivos por comparación de datos. Un sistema relacional tiene la flexibilidad de tomar uno o dos archivos y generar un nuevo archivo con los registros que coinciden con un criterio específico.

BMP
Bean Managed Persistence. Tipo de Entity Bean cuya persistencia es manejada por el propio Bean.

Browser
Ver Navegador.

❖ C

Caso de Uso
Descripción narrativa textual de la secuencia de eventos y acciones que ocurren cuando un usuario se encuentra en diálogo con un sistema durante un proceso significativo.

Caso de Uso de Negocio
Describe un proceso del negocio que tiene como consecuencia la utilización del sistema, pero no se limita solamente a lo que se refiere a la interacción con el sistema. Su objetivo es proporcionar una visión del entorno en el cual se llevan a cabo los casos de uso del sistema, para permitir una mejor comprensión de los mismos. Generalmente, un caso de uso de negocio desencadena en uno o más casos de uso del sistema.

Caso de Uso de Sistema
Ver Caso de Uso.

CBD
Forma abreviada de referirse a la metodología de desarrollo basado en componentes descripta en [VP03] y utilizada en este proyecto.

Cliente
Aplicación o proceso que solicita un servicio de algún proceso o componente. Un cliente facilita una conexión con un servidor, administra y presenta la información recuperada del mismo. En un entorno cliente-servidor, la estación de trabajo es normalmente el equipo cliente.

CMP
Container Managed Persistence. Tipo de Entity Bean cuya persistencia es manejada por el contenedor J2EE.

CORBA
Common Object Request Broker Architecture. Estándar del Object Management Group (OMG) que ofrece la tecnología necesaria para sistemas distribuidos, define un lenguaje IDL (Interface Definition Language), utiliza Object Request Brokers (ORBs) para facilitar la comunicación entre los espacios de objetos y define una serie de servicios generalmente ventajosos.

❖ D

DHTML
Dinamic HTML. Aplicaciones que contienen objetos y eventos y se procesan en el lado del cliente dentro del navegador Web.

Diseño
Proceso que se sirve de los productos del análisis para generar una especificación destinada a implementar un sistema. Descripción lógica de cómo funcionará el sistema.

Diseño orientado a objetos
Especificación de una solución lógica de software a partir de objetos de software: clases, atributos, métodos y colaboraciones.

❖ E

Enterprise JavaBean
Componente del lado del servidor que habilita y simplifica el proceso de construir una aplicación empresarial distribuida en java. Permite que los programadores se enfoquen en su aplicación y no en la infraestructura de la misma. Típicamente contienen la lógica del negocio (SessionBeans) y mantienen vivos los datos de dicho negocio (EntityBeans).

EJB
Forma abreviada de referirse a los Enterprise JavaBeans de J2EE.

❖ F

Framework
Término utilizado en programación orientada a objetos para definir un conjunto de clases e interfaces que definen un diseño abstracto para solucionar un grupo de problemas relacionados.

❖ G

GRASP
General Responsability Asignment Software Patterns. Patrones generales de software para asignar responsabilidades planteados en [Lar99].

❖ H

HTML
Hyper Text Markup Language (Lenguaje de marcado de hypertexto). Es el lenguaje estándar de la Web. Está compuesto de etiquetas (tags) que indican al cliente Web (browser) cómo mostrar el contenido de un documento.

HTTP
HyperText Transport Protocol. Protocolo de comunicaciones utilizado para conectarse a servidores en la Web. Su función primaria es establecer una conexión con un servidor Web y transmitir páginas HTML al navegador.

❖ I

IBM
International Business Machines Corporation. Compañía que brinda productos y servicios para empresas. Investiga, desarrolla y fabrica sistemas informáticos, software, redes, sistemas de almacenamiento y microelectrónica.

Internet
Es un sistema mundial de redes de computadoras, un conjunto integrado por las diferentes redes de cada país del mundo, por medio del cual un usuario en cualquier computadora puede, en caso de contar con los permisos apropiados, acceder a información de otra computadora e incluso tener comunicación directa con otros usuarios del mundo.

ISA
Forma abreviada de referirse a la empresa donde se desarrolló el presente proyecto (Ver ISA Ltda.).

ISA Ltda.
Ingenieros de Sistemas Asociados Ltda. Empresa en la que se desarrolló el presente proyecto. Fundada en octubre de 1990, prestadora de servicios de consultoría en informática, desarrollo de sistemas e ingeniería.

❖ J

J2EE
Java 2 Enterprise Edition. Especificación abierta diseñada por SUN Microsystems, junto con otros socios de la industria, que permite desarrollar aplicaciones del lado del servidor.

Java
Lenguaje de programación, derivado de C++, desarrollado por SUN Microsystems que se ofrece como un estándar abierto.

JavaBean
Componente Java reusable, el cual tiene una interfaz estandarizada (constructor sin argumentos, propiedades, métodos y eventos).

JavaScript
Lenguaje de scripting desarrollado por Netscape Corporation, el cual permite que diseñadores de páginas Web puedan crear contenido interactivo en ellas.

JavaServer Page
Documento XML o HTML que tiene algunos tags especiales. Permite realizar la presentación de contenido dinámico en forma separada de la lógica del negocio. En el momento de ejecución se transforma en un Servlet.

JDBC
Java Database Connectivity. API que permite conectarse con una base de datos a través de un driver y ejecutar sentencias SQL en ella.

JDO
Java Data Objects. API que brinda la abstracción del manejo de persistencia de datos; desarrolladores Java que no tengan conocimiento previo de la base de datos o lenguaje de consulta pueden realizar las conexiones utilizando esta API.

JNDI
Java Naming and Directory Interface. API que permite trabajar con servicios de nombramiento (Naming) y directorio (Directory). El servicio de nombramiento asocia nombres con objetos (por ejemplo DNS). El servicio de directorio también asocia nombres con objetos pero provee información adicional a través de atributos (por ejemplo LDAP).

JSP
Forma abreviada de referirse a una JavaServer Page.

❖ M

Metodología de desarrollo basado en componentes
Metodología de desarrollo, presentada en [VP03], orientada a la construcción de aplicaciones empresariales de gran porte. Busca la producción de software de buena calidad basándose en la utilización de componentes que sean fácilmente reemplazables en caso de ser necesario.

Middleware
Aplicaciones especiales que, interpuestas entre otras dos aplicaciones, permiten que éstas se comuniquen entre sí, a pesar de utilizar protocolos, arquitecturas o sistemas operativos diferentes.

❖ N

Navegador
Software utilizado para localizar y desplegar páginas Web. Permite desplegar gráficos, texto e información multimedia.

❖ P

Patrón
Es la descripción etiquetada de un problema, de la solución, de cuándo aplicar la solución y la manera de hacerlo dentro de otros contextos.

Patrón de diseño
Un patrón de diseño describe una solución probada para un problema recurrente a la hora de diseñar aplicaciones de software.

Prototipo
Versión de un sistema de computación que contiene algunas características básicas del sistema real, aunque no necesariamente las mismas características técnicas ni toda la funcionalidad.

❖ R

RUP
Rational Unified Process. Producto de Rational Software que presenta un modelo de proceso de ingeniería de software completo.

❖ S

Servidor
Equipo de una red que ejecuta aplicaciones para otros equipos de ella. También se denomina de esta forma al software que se ejecuta en dicho equipo, efectuando la tarea de servir archivos y ejecutar aplicaciones.

Servlet
Clase Java que implementa un componente Web que corre del lado del servidor y que es manejado por el contenedor Web. Se utiliza para generar contenido dinámico.

SGC
Sistema de Gestión Comercial tomado como caso de estudio de este proyecto.

Sistemas legados
Son sistemas operacionales ya existentes en las empresas a la hora de crear un nuevo sistema. Dichos sistemas contienen información de gran valor, la cual debe mantenerse.

Struts
Forma reducida de referirse al framework Apache Struts.

❖ U

UML
Unified Modeling Language (lenguaje unificado de modelamiento). Lenguaje de modelado visual que se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se usa para entender, diseñar, configurar, mantener y controlar la información de los sistemas a construir.

URL
Uniform Resource Location (localizador uniforme de recurso). Son las direcciones que se usan en la Web para localizar determinado recurso (página).

❖ W

Web
World Wide Web. Universo de información accesible a través de Internet.

WebSphere Studio Application Developer
Entorno de desarrollo integrado, utilizado en este proyecto, que forma parte de los productos de la familia WebSphere Studio de IBM. Por mayor información dirigirse a [WF].

WSAD
Forma reducida de hacer referencia a la herramienta WebSphere Studio Application Developer.

❖ X

XML
Extensible Markup Language. Conjunto de reglas para definir etiquetas semánticas que nos organizan un documento en diferentes partes. XML es un metalenguaje que define la sintaxis utilizada para definir otros lenguajes de etiquetas estructurados.

----- ¤ -----

Lista de Anexos

A continuación se detallan los títulos de cada uno de los anexos incluidos en los dos documentos presentados junto con este informe. En la sección 1.4 se puede encontrar una breve descripción sobre el contenido de cada uno de ellos.

Anexos Teóricos:

Anexo 1. Introducción al Análisis y Diseño Orientado a Objetos
Anexo 2. Proceso de Identificación de Requerimientos
Anexo 3. Construcción del Modelo Conceptual
Anexo 4. Patrones GRASP
Anexo 5. Evolución de las aplicaciones Web
Anexo 6. Análisis de Tecnologías J2EE & .NET
Anexo 7. Desafíos de aplicaciones empresariales
Anexo 8. Arquitectura J2EE
Anexo 9. Servlets
Anexo 10. JSPs
Anexo 11. JavaBeans
Anexo 12. EJBs
Anexo 13. Patrones de Diseño J2EE
Anexo 14. Frameworks
Anexo 15. Struts

Anexos Particulares:

Anexo 16. Especificación de Requerimientos
Anexo 17. Casos de Uso
Anexo 18. Modelo Conceptual
Anexo 19. Caso de Estudio: Análisis y Diseño
Anexo 20. Detalles de la implementación en WebSphere
Anexo 21. Actas de Reuniones
Anexo 22. Historia de la empresa ISA Ltda.

----- ¤ -----