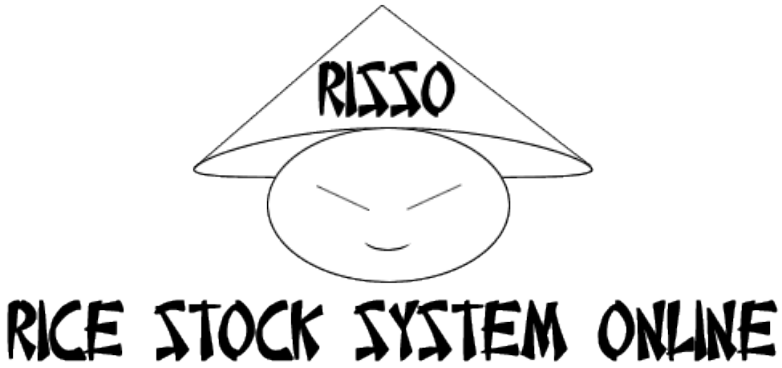
# IMPLEMENTACIÓN DE WEB SERVICES EN PLATAFORMA J2EE PARA UNA EMPRESA TRILLADORA DE ARROZ




**ANA MARÍA AMAYA ROJAS**
**MÓNICA ALEJANDRA VEGA URIBE**

**UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA**
**ESCUELA DE CIENCIAS NATURALES E INGENIERÍA**
**FACULTAD DE INGENIERÍA DE SISTEMAS**
**LÍNEA DE SISTEMAS DE INFORMACIÓN E INGENIERÍA DE SOFTWARE**
**BUCARAMANGA**
**2006**

# IMPLEMENTACIÓN DE WEB SERVICES EN PLATAFORMA J2EE PARA UNA EMPRESA TRILLADORA DE ARROZ

**Autores:**
**ANA MARÍA AMAYA ROJAS**
**MÓNICA ALEJANDRA VEGA URIBE**

**Trabajo de Grado presentado como requisito parcial para optar por el título de Ingeniero de Sistemas.**

**Directora:**
**MCC (c) GARETH BARRERA SANABRIA**

**UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA**
**ESCUELA DE CIENCIAS NATURALES E INGENIERÍA**
**FACULTAD DE INGENIERÍA DE SISTEMAS**
**LÍNEA DE SISTEMAS DE INFORMACIÓN E INGENIERÍA DE SOFTWARE**
**BUCARAMANGA**
**2006**

**Nota de aceptación**

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Bucaramanga, mayo 19 de 2006

## DEDICATORIA

A mis padres, Luis Alfonso y Claudia Helena, quienes siempre me han brindado su apoyo y amor.

ANA MARÍA

A mis padres, Antonio Vega y Emilce Uribe, por darme su apoyo durante todo el proceso de mi formación profesional, gracias por estar siempre a mi lado ayudándome en los momentos difíciles y por poner en mí toda su confianza y afecto.

A mis hermanos, Diana Vega y Jesús A. Vega, que han sido de ayuda en mi formación como ser humano y profesional, gracias por creer en mí y en mis capacidades.

MÓNICA ALEJANDRA

## AGRADECIMIENTOS

Los autores agradecen a Dios por el don de la vida y la oportunidad de encontrarnos con las siguientes personas que con sus aportes fueron dando una luz para cumplir con los objetivos inicialmente planteados:

- Daniel Arenas, por enseñarnos sobre la Ingeniería de Software.
- Sandra Moreno, por sus valiosas correcciones sobre RUP y UML.
- Freddy Méndez, por sus compartir sus conocimientos de la Tecnología Java.
- Edgar López, por facilitarnos el libro electrónico "Web Services: A manager's guide" de Anne Thomas Mannes.
- John Rangel, por facilitarnos el libro electrónico “JBoss at Work: A Practical Guide” de Scout Davis y Tom Marrs.
- Carlos Harker, Administrador de “Arroz San Cristóbal”, quien nos brindó información clave para el modelado del negocio de la empresa trilladora de arroz.

Finalmente, y no menos importante, a la coordinadora y a la secretaria de proyectos de la Facultad de Ingeniería de Sistemas, Gareth Barrera y Helga Mora, respectivamente; así como a las personas de otras dependencias de la universidad como el Departamento de Servicios Computacionales, biblioteca, cafetería, vigilancia y parqueadero.

**¡MUCHAS GRACIAS!**

**CONTENIDO**

## LISTA DE FIGURAS

## RESUMEN

En las empresas trilladoras de arroz se compra paddy, o grano verde, que debe ser sometido a distintos procesos de transformación hasta obtener un producto de alta calidad para la venta al consumidor. En cada proceso las cantidades del grano deben ser tomadas para estimar el rendimiento del insumo de cada cultivador y registradas en el stock, o nivel de existencias, para que el vendedor pueda ofrecer tanto los productos terminados como los no terminados. Por lo general estos registros se llevan en formatos de papel que luego son ingresados a hojas de cálculo, lo cual hace que el proceso de almacenamiento de datos no se encuentre en tiempo real y el registro de éstos no sea tan necesario. Además al no encontrarse disponibles los datos en la red se dificultaría el proceso de venta en otros puntos diferentes al sitio de producción.

Por consiguiente se desarrollará un prototipo de sistema de información en línea con el cual se podrá registrar el inventario del grano en sus procesos de transformación y la facturación de compras de insumos y ventas de productos en las empresas trilladoras en general, mientras en tiempo de ejecución notifica a un paquete de contabilidad sobre las cuentas por pagar (liquidaciones) y las cuentas por cobrar (facturas). Con todo lo anterior se empleará la plataforma **J2EE**, con tecnologías **JSP**, **EJB** y **WS** para reducir todos estos problemas de integración de datos ya que la implementación de éstas tecnologías ayudará a exponer las funcionalidades de una aplicación, la cual podrá ser accesible desde Internet, para así obtener una aplicación con arquitectura multinivel, que promueva la modularidad y escalabilidad.

# INTRODUCCIÓN

El desarrollo de aplicaciones se hace de acuerdo a las necesidades del cliente y a las plataformas que ofrece el mercado de la Ingeniería de Software a los desarrolladores. Generalmente los sistemas de información empresariales han sido diseñados para cubrir una necesidad específica, como la contabilidad de la empresa, pero ahora el enfoque está en diseñar aplicaciones integrales que abarquen desde la nómina, hasta la facturación y la contabilidad o integrar las aplicaciones que ya existen y cumplen esas tareas por separado. La tarea de integrar abarca una serie de soluciones denominadas *middleware*, al ubicarse en medio de las aplicaciones y los sistemas operativos; y han evolucionado desde los sockets, pasando por RPC y ahora los *Web Services*.

Para nuestro caso, se diseñó una aplicación de inventario para las empresas trilladoras de arroz que se puede comunicar con un paquete de contabilidad a través de la Internet, para llevar registros de las cantidades del grano desde la compra de insumos hasta la venta de productos y de sus respectivas cuentas contables, por cobrar y por pagar.

Ésta aplicación se desarrolló empleando el Lenguaje Java por su esquema de Programación Orientada a Objetos y su lema de WORA (Write Once, Run Anywhere), el cual implica que un programa puede ser desarrollado en cualquier dispositivo, compilado con un JDK (Java Development Kit) y ejecutado desde cualquier equipo con la JVM (Java Virtual Machine) así como la tecnología de los Web Services con su esquema de Arquitectura Orientada a Servicios. Se eligió seguir la metodología de desarrollo de software que reúne las características de las precedentes y por lo tanto la hace más completa, es conocida como RUP

(Rational Unified Process) y se complementa con UML (Unified Modeling Language), para los diagramas de diseño.

Es así como el resultado del presente trabajo de grado es un prototipo altamente escalable y reutilizable, por su naturaleza multicapa, por manejar un esquema en general para una línea de empresas con una lógica del negocio en común, por emplear un lenguaje de extenso uso y por emplear herramientas libres de licencias lo cual arroja unos componentes genéricos que promueven un posterior mantenimiento eficiente o un redesarrollo con menores cantidades de tiempo y costos de desarrollo. De manera que se empleó todo un marco de desarrollo de trabajo como lo plantean las mejores prácticas de Ingeniería de Software, siguiendo una metodología y basados en una plataforma como J2EE.

A continuación se detallan cada una de las tecnologías empleadas en el presente trabajo de grado, así como las herramientas empleadas y los entregables obtenidos en cada fase del proceso de desarrollo de software. Cabe destacar que cada capítulo contiene información tanto teórica como práctica lo cual se presenta organizado de la siguiente manera:

Capitulo 1, contiene los antecedentes que dieron paso al inicio de la planeación del presente trabajo de grado y a la elección de los paradigmas de programación que se usaron en el desarrollo del prototipo.

Capítulo 2, contiene los temas más relevantes de los Web Services como los antecedentes, definición, características, estándares, funcionamiento, estado del arte y protocolos de transporte.

Capítulo 3, contiene información de J2EE como arquitectura de software que soporta las acciones del prototipo con sus APIs y niveles de arquitectura empresarial.

Capítulo 4, contiene un resumen de la metodología aplicada a lo largo del proyecto, RUP con UML.

Se finaliza el documento con las conclusiones y las recomendaciones; así como con la documentación del prototipo (plantillas) siguiendo las fases de la metodología anteriormente mencionada.

# 1. GENERALIDADES DEL PROTOTIPO

## 1.1. ANTECEDENTES

En las empresas comercializadoras de arroz se compra producto de diferentes características los cuales deben ser sometidos a diferentes procesos, compra, limpieza, secado, trillado, empaque y venta, para de esta manera obtener un producto óptimo para satisfacer las necesidades del consumidor. En cada estado o transformación las cantidades del producto deben ser controladas de tal manera que se conozcan los costos y se estimen las cantidades a obtener en la siguiente transformación.

Hoy día los comercializadores de arroz conocen los inventarios de entrada y solo hasta el final saben los márgenes de utilidad que h an obtenido. El gerente comercial a través de indicadores del mercado podría estimar qué cantidad de arroz verde debe comprar para cumplir con las metas de ventas. Integrar las operaciones de compra y venta que se realizan durante el día en las diferentes sucursales o por los diferentes vendedores requiere de personal adicional y tiempos en los cuales los datos proporcionados por los procesos son registrados de manera manual y luego almacenados en hojas de cálculo, provocando que no se haga un seguimiento detallado al rendimiento y calidad del insumo, así mismo la información suministrada no es suficiente para estimar la fecha en que habrá nuevos productos generando que el inventario final no se encuentre al día con la contabilidad de la empresa; es por esto que la comunicación al departamento de contabilidad se realiza días después por lo cual la alta dirección no podrá tomar decisiones importantes de acuerdo a los movimientos de oferta y demanda del mercado.

Es por esto que adoptar procesos automatizados de información ha sido la alternativa más común en las empresas para disminuir los inconvenientes de comunicación entre las diferentes áreas. Los procesos manuales de crear, modificar o eliminar son lentas e inseguras, y cuando se utiliza Excel se corre el riesgo de perder la información, ya que se encuentra aislada de otras aplicaciones haciéndola difícil de mantener actualizada, presentando, en algunas ecuaciones, redundancia de datos por la ausencia de un sistema centralizado.

## 1.2. PARADIGMAS DE PROGRAMACIÓN

Los Paradigmas de Programación constituyen un conjunto de modelos conceptuales que colaboran con el desarrollo del proceso de diseño estableciendo la estructura de un programa. Sin duda alguna hoy por hoy existen una gran variedad de Paradigmas de Programación tales como, Paradigma Imperativo, Paradigma Funcional, Paradigma Lógico, entre otros; sin embargo algunos programadores se centran en un único paradigma el cual, para ellos, es el que mejor satisface sus necesidades al momento de desarrollar software. Para el desarrollo de nuestro prototipo hemos aplicado el Paradigma de Orientación a Objetos (POO) y el Paradigma o Arquitectura Orientada a Servicios (SOA, Service Oriented Architecture), los cuales se explicarán a continuación.

### 1.2.1. OOP (Object Oriented Programming)

El término de Programación Orientada a Objetos (POO) no es nada nuevo en el mundo del desarrollo de software debido que programas como *SmallTalk*, C++, Visual Basic, entre otros, se encuentran basados en ella.

La Programación Orientada a Objetos es un tipo de programación basada en clases de objetos los cuales cambian de estado, comportamiento e identidad. Los programas son implementados como un conjunto de éstos objetos los cuales contribuyen entre sí para realizar alguna tarea específica. El principal elemento en este tipo de programación es el *objeto*, el cual mediante sus atributos permite ser

definido e identificado de los demás objetos incluidos en una su misma clase o en objetos pertenecientes a otras clases.

A continuación se enunciarán algunas características de la Programación Orientada a Objetos:

- Abstracción: Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar *cómo* se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.

- Encapsulamiento: También llamado "ocultación de la información". Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una *interfaz* a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

- Polimorfismo: Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado.

Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama *asignación tardía* o *asignación dinámica*. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

- Herencia: Las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y la encapsulación permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que reimplementar su comportamiento. Esto suele hacerse habitualmente agrupando los objetos en *clases* y estas en *árboles* o *enrejados* que reflejan un comportamiento común. Cuando un objeto pertenece a más de una clase se llama herencia múltiple; esta característica no está soportada por algunos lenguajes (como Java).

- Tipificación: Permite la agrupación de objetos en tipos.

- Concurrencia: Los objetos pueden actuar al mismo tiempo.

- Persistencia: Un objeto puede seguir existiendo tras desaparecer su antecesor[1].

Aparte de las características mencionadas también hay que considerar las ventajas y problemas que se obtienen al implementar este tipo de paradigma.

[1] WIKIPEDIA, La Enciclopedia Libre. Programación Orientada a Objetos. [En línea]. Fecha de Modificación: Abril 2 de 2006. Disponible en Internet <URL: http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos>

Ventajas:

- Uniformidad: Ya que la representación de los objetos lleva implica tanto el análisis como el diseño y la codificación de los mismos.

- Comprensión: Tanto los datos que componen los objetos, como los procedimientos que los manipulan, están agrupados en clases, que se corresponden con las estructuras de información que el programa trata.

- Flexibilidad: Al tener relacionados los procedimientos que manipulan los datos con los datos a tratar, cualquier cambio que se realice sobre ellos quedará reflejado automáticamente en cualquier lugar donde estos datos aparezcan.

- Estabilidad: Dado que permite un tratamiento diferenciado de aquellos objetos que permanecen constantes en el tiempo sobre aquellos que cambian con frecuencia permite aislar las partes del programa que permanecen inalterables en el tiempo.

- Reusabilidad: La noción de objeto permite que programas que traten las mismas estructuras de información reutilicen las definiciones de objetos empleadas en otros programas e incluso los procedimientos que los manipulan. De esta forma, el desarrollo de un programa puede llegar a ser una simple combinación de objetos ya definidos donde estos están relacionados de una manera particular[2].

Problemas:

- Curvas de aprendizaje largas: Un sistema orientado a objetos ve al mundo en una forma única. Involucra la conceptualización de todos los elementos de un programa, desde subsistemas a los datos, en la forma de objetos. Toda la comunicación entre los objetos debe realizarse en la forma de mensajes. Esta no es la forma en que están escritos los programas orientados a objetos

[2] MARTIN, Tejarina. Programación Orientada a Objetos–(OOP, Object Oriented Programming). [En línea]. Fecha de Publicación Septiembre 25 de 2003. Disponible en Interne <URL:http://www.ilustrados.com/publicaciones/EpyuVuuAlEAFjVSJEI.php>

actualmente; al hacer la transición a un sistema orientado a objetos la mayoría de los programadores deben capacitarse nuevamente antes de poder usarlo.

- Dependencia del lenguaje: A pesar de la portabilidad conceptual de los objetos en un sistema orientado a objetos, en la práctica existen muchas dependencias. Muchos lenguajes orientados a objetos están compitiendo actualmente para dominar el mercado. Cambiar el lenguaje de implementación de un sistema orientado a objetos no es una tarea sencilla; por ejemplo C++ soporta el concepto de herencia múltiple mientras que *SmallTalk* no lo soporta; en consecuencia la elección de un lenguaje tiene ramificaciones de diseño muy importantes.

- Determinación de las clases: Una clase es un molde que se utiliza para crear nuevos objetos. En consecuencia es importante crear el conjunto de clases adecuado para un proyecto. Desafortunadamente la definición de las clases es más un arte que una ciencia. Si bien hay muchas jerarquías de clase predefinidas usualmente se deben crear clases específicas para la aplicación que se este desarrollando. Luego, en 6 meses ó 1 año se da cuenta que las clases que se establecieron no son posibles; en ese caso será necesario reestructurar la jerarquía de clases devastando totalmente la planificación original.

- Performance: En un sistema donde **todo** es un objeto y **toda** interacción es a través de mensajes, el tráfico de mensajes afecta la performance. A medida que la tecnología avanza y la velocidad de microprocesamiento, potencia y tamaño de la memoria aumentan, la situación mejorará; pero en la situación actual, un diseño de una aplicación orientada a objetos que no tiene en cuenta la performance no será viable comercialmente[3].

[3] MOREA, Lucas. Sistemas de Procesamiento de Datos. Programación Orientada a Objetos. [En línea]. Fecha de Publicación: 1997. Disponible en Internet <URL: http://www.monografias.com/trabajos/objetos/objetos.shtml>

### 1.2.2. SOA (Service-Oriented Architecture)

Arquitectura Orientada a Servicios es un conjunto de prácticas en común para aplicaciones basadas en servicios que puedan describirse a sí mismos, ser publicados y encontrados, para entregar servicios conforme sean requeridos. SOA no implica WS (Web Service), y WS no implica SOA, pero WS es la manera más común de aplicar SOA.

La Programación Orientada a Servicios o mejor conocida como la Arquitectura Orientada a Servicios es un complemento de la Programación Orientada a Objetos. La Arquitectura Orientada a Servicios es un modelo de diseño que guarda la lógica de la aplicación dentro de servicios los cuales interactúan a través de los protocolos de comunicación; la cual surge como consecuencia de la necesidad de otorgar más flexibilidad en las aplicaciones para hacer más fácil el cambio de los procesos de acuerdo a las situaciones que se presenten en las empresas.

Seguidamente presentaremos algunos beneficios que caracterizan este tipo de arquitectura:

- En las aplicaciones SOA se desarrollan componentes reutilizables fáciles de mantener y probar.
- Para cada aplicación se reutilizan componentes que ya existan haciendo que sólo se desarrollen aquellos que sean necesarios.

Las Arquitecturas Orientadas a Servicios están siendo puestas de moda por la incursión de los Servicios Web en el mundo de las aplicaciones distribuidas, teniendo en cuenta esto SOA presenta las siguientes propiedades:

- Vista Lógica: El servicio es una abstracción (vista lógica) de los programas, bases de datos, procesos de negocio, etc., definido en términos de lo que hace (llevando a cabo una operación de negocio).

- Orientación a Mensaje: El servicio se define formalmente en términos de los mensajes intercambiados entre agentes y solicitantes, y no está basado en las propiedades de los agentes. La estructura interna del agente (lenguaje de programación, BD, proceso, etc.) se abstrae en SOA. Esto permite incorporar cualquier componente o aplicación a esta arquitectura "decorando" estos componentes con software de gestión y conversión.
- Orientación a la Descripción: Un servicio se describe con metadatos procesables. La descripción da soporte a la naturaleza pública de SOA: sólo se incluyen en la descripción aquellos detalles que se exponen públicamente y son importantes para el uso del servicio. La semántica de un servicio debe documentarse, directa o indirectamente, por su descripción.
- Granularidad: Los servicios tienden a usar un pequeño número de operaciones con mensajes relativamente complejos.
- Orientación a la Red: Los servicios tienden a usarse a través de la red, aunque esto no es un requisito absoluto.
- Neutral a la Plataforma: Los mensajes se envían en un formato estándar y neutral a la plataforma, distribuido a través de los interfaces (XML)[4].

[4] PELACHANO, Vicente. Servicios Web. Estándares, Extensiones y Perspectivas de Futuro. Universidad Pontificia Politécnica de Valencia. Departamento de Sistemas Informáticos y Computación.

# 2. WEB SERVICES

## 2.1. DEFINICIÓN

Servicio Web es un componente que contiene lógica de negocios y presta servicios a través de una interfaz, cumple con las propiedades de SOA y se clasifica como una solución de *middleware* que permite integración no intrusiva entre aplicaciones heterogéneas y con bajo costo de desarrollo y mantenimiento, así como interoperabilidad debido al empleo estándares abiertos.

Inicialmente el WS es creado y desplegado por un archivo WSDD, luego descrito por un archivo WSDL, publicado en un repositorio UDDI, transportado por mensajes SOAP con contenido XML sobre HTTP y finalmente consumido por una aplicación cliente, o stub, que crea una conexión proxy para acceder al servicio. Está diseñado para ser accedido por aplicaciones a través de la web y es clave para el futuro de la segunda generación de la web o web semántica. Existen dos tipos de servicios web: sincrónicos (RPC) y asincrónicos (MOM). Sus especificaciones son administradas por las organizaciones OASIS, W3C y WS-I.

## 2.2. CARACTERÍSTICAS

- Encapsulamiento: La lógica de servicios debe ser accedida sólo a través de una interfaz pública bien definida.
- Acoplamiento débil: Su ejecución no depende de la interacción con componentes externos.
- Construcción con base en componentes: Los servicios son una combinación de componentes de negocio. Su lógica de procesamiento se debe limitar a instanciar los componentes y a tomar decisiones simples dependiendo de los resultados de los métodos invocados.

- Enfoque en procesos: La combinación de los componentes utilizados por un servicio debe representar un proceso.
- Orientación a uso en la red: Un servicio debe tener una interfaz accesible desde la red. Para nuestra tecnología los servicios deben exponerse como componentes límite. En el momento de adquirir la infraestructura adecuada, los servicios podrán exponerse como EJBs, WebServices, etc.
- Reutilización: Los servicios deben ser diseñados para ser reutilizables, evitando la duplicidad de implementación de la misma lógica.
- Construcción con base en estándares: Disminuye el tiempo de mantenimiento y hace más fácil la capacitación en su funcionamiento, además fortalece la reutilización. [5]

## 2.3. ESTÁNDARES

Para que se haga efectivo el proceso de comunicación entre dos aplicaciones mediante el uso de los *web services* son necesarias cuatro tecnologías propias de éste middleware: la primera que se encargue de codificar los mensajes que serán transmitidos entre los equipos; la segunda que resuma los métodos, parámetros y ubicación de la interfaz; la tercera que se encargue de agregarle información significativa al mensaje sobre el emisor y el remitente y la cuarta que hace una clasificación según la taxonomía.

### 2.3.1. XML (eXtended Markup Language)

El Lenguaje de Marcado Extendido ha sido diseñado por la W3C para crear lenguajes según la necesidad del negocio y soporten de una mejor manera los servicios que ofrece la Internet, como páginas estáticas y dinámicas. En base a éste lenguaje se crearon las etiquetas de las siguientes tecnologías: WSDL, SOAP y UDDI. El uso de este metalenguaje proporciona la característica a los web services de que sean independientes del lenguaje y de la plataforma.

---

[5] BUSTAMANTE, Andrés Camilo. Implementación de una visión de arquitectura: Experiencias y Resultados. En: Salón de Informática. Arquitecturas Empresariales de Software: Espejismos y Realidades. (7°: 2005: Bogotá). Memorias del Salón de Informática. Bogotá, Asociación Colombiana de Ingenieros de Sistemas, 2005.

### 2.3.2. WSDL (Web Services Description Language)

El Lenguaje de Descripción de Servicios Web se usa para crear un contrato donde se describe qué métodos tiene el *Web Service*, cuáles son sus parámetros de entrada y salida, así como el protocolo de transporte elegido y la URL donde puede ser accesado. Este archivo con extensión .wsdl, tiene etiquetas XML que permiten a los clientes generar conexiones dinámicas para poder consumir el web service. Aunque herramientas como Apache Axis o IDEs como NetBeans pueden generar este archivo en base a clases .java o .jws, es necesario comprender su contenido para su posterior publicación y comprensión, así como se puede apreciar en la figura 1.

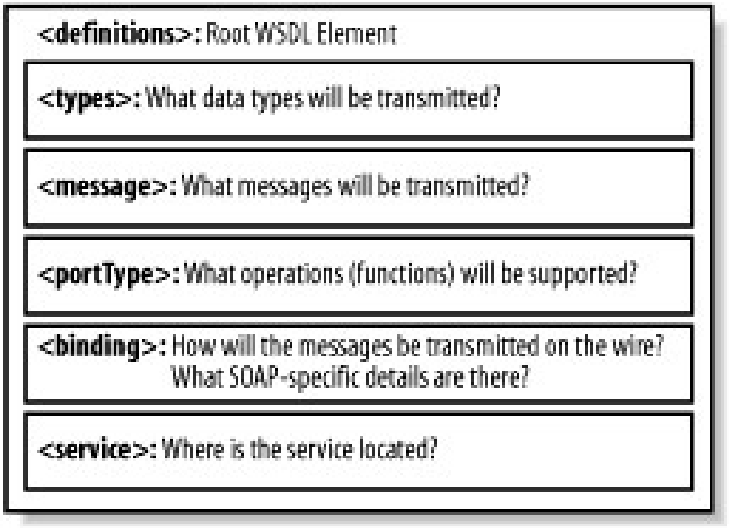



Figura 1. Etiquetas del Lenguaje WSDL [6]

El archivo inicia con la definición de XML, así como la versión y su codificación. Luego viene la etiqueta principal *definitions* cuyos atributos son el nombre del servicio y los espacios de nombre de las diferentes tecnologías que se usarán en el resto del archivo como WSDL, SOAP y XSD. Luego aparecen dos etiquetas *message* por cada método del servicio, uno para solicitud y otro para respuesta, *Request* y *Response* respectivamente; de ahí se desprende otra etiqueta llamada

[6] CERAMI, Ethan. Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. Sebastopol: O'Reilly, 2002. P. 121.

*part* con un nombre y un tipo de dato. Continúa en la jerarquía del archivo la etiqueta *portType* con una subetiqueta, llamada *operation*, por cada método del servicio; de la cual surgen las etiquetas *input* y *output* que hacen referencia al contenido del atributo *name* en la etiqueta *message* descritos anteriormente. Sigue la etiqueta *binding* que relaciona los métodos de entrada y salida definidos en *portType* dentro de la subetiqueta *operation* para el protocolo de transporte SOAP, de nuevo una por cada método del servicio. Finalmente la etiqueta *service* con subetiquetas *documentation* y *port* que con sus atributos proporcionan el nombre y la ubicación del archivo WSDL.

### 2.3.3. SOAP (Simple Object Access Protocol)

Este protocolo funciona como un sobre para los mensajes XML que se transmitirán entre equipos remotos, por lo tanto se compara con un sobre que viajará sobre el protocolo de transporte elegido para el intercambio de paquetes. Así como HTTP, existe una solicitud y una respuesta y cada uno comprende de elementos y atributos que pueden ser obligatorios u opcionales. El elemento principal *Envelope* tiene los espacios de nombre que se usarán a lo largo del mensaje, así como un elemento *Header* (opcional) que tiene información de enrutamiento y/o seguridad y *Body* (obligatorio) que contiene el mensaje y/o archivos adjuntos. Sin embargo como ésta sintaxis no es necesaria comprenderla, no se entrará en detalles. También existe otro protocolo de acceso que funciona de manera similar a SOAP, se llama XML-RPC pero éste no se ha convertido en un estándar como sí ocurrió con SOAP 1.2 el 24 de junio del año 2003 por el grupo *XML Protocol Working Group* de la W3C.

### 2.3.4. UDDI (Universal Description, Discovery and Integration)

Es un servicio web que ha sido estandarizado por OASIS, y presta sus servicios como repositorio para los servicios descritos por medio de un archivo WSDL, donde los proveedores de *Web Services* pueden publicar información de sus servicios según una taxonomía para que luego los clientes puedan ubicarlos como en un directorio telefónico así:

- Páginas blancas – para buscar una empresa (naming).
- Páginas amarillas – para buscar la descripción de un servicio (trading).
- Páginas verdes – para buscar un proveedor de servicio (registro). [7]

2.3.5. ebXML (electronic business using XML)

Negocios electrónicos usando el Lenguaje de Marcado Extensible es un *framework* de *Web Services* diseñado para soportar las transacciones de entre negocios (B2B) y fue desarrollado en conjunto por OASIS y UN/CEFACT. Tiene protocolos paralelos a los mencionados anteriormente así:

- ebMS (ebXML Message Service): Servicio de Mensajes que extiende SOAP agregándole adjuntos y seguridad. Paralelo a SOAP.
- CPPA (Collaboration Protocol Profile and Agreement): Perfil y Acuerdo del Protocolo de Colaboración, lenguaje XML para especificar detalles de cómo una compañía soporta integración B2B. Paralelo a WSDL.
- BPSS (Business Process Specification Schema): Esquema de la Especificación del Proceso del Negocio, un lenguaje XML usado para describir el intercambio de mensajes que deben ser intercambiados para completar la transacción de negocios especificada. Paralelo a XSLT.
- ebXML Registry and Repository: Registro y Depósito, administra la información acerca de los tipos y proveedores de servicio, así como un repositorio para las descripciones de servicios, esquemas, descripciones CPPA, especificaciones BPSS, y otros metadatos. Paralelo a UDDI.[8]

Cabe anotar que éstos protocolos mencionados anteriormente funcionan a la par que los protocolos WSDL/SOAP/UDDI, pero la industria de los desarrolladores ya que éstos fueron declarados como estándares por la W3C, máxima autoridad de la WWW.

[7] WIKIPEDIA, La enciclopedia libre. UDDI. [En línea]. Disponible en Internet <URL:http://es.wikipedia.org/wiki/UDDI>

[8] THOMAS MANES, Anne. Web Services: A manager's guide. Boston: Addison-Wesley, 2003. 352 p.

## 2.4. FUNCIONAMIENTO

Es ahora cuando se muestra de qué manera se relacionan los tres estándares mencionados anteriormente (WSDL, SOAP, UDDI), con los tres roles (Proveedor, Distribuidor, Consumidor) y con las tres operaciones (Publicar, Buscar, Unir) que se describirán posteriormente y se pueden apreciar en la figura 2.

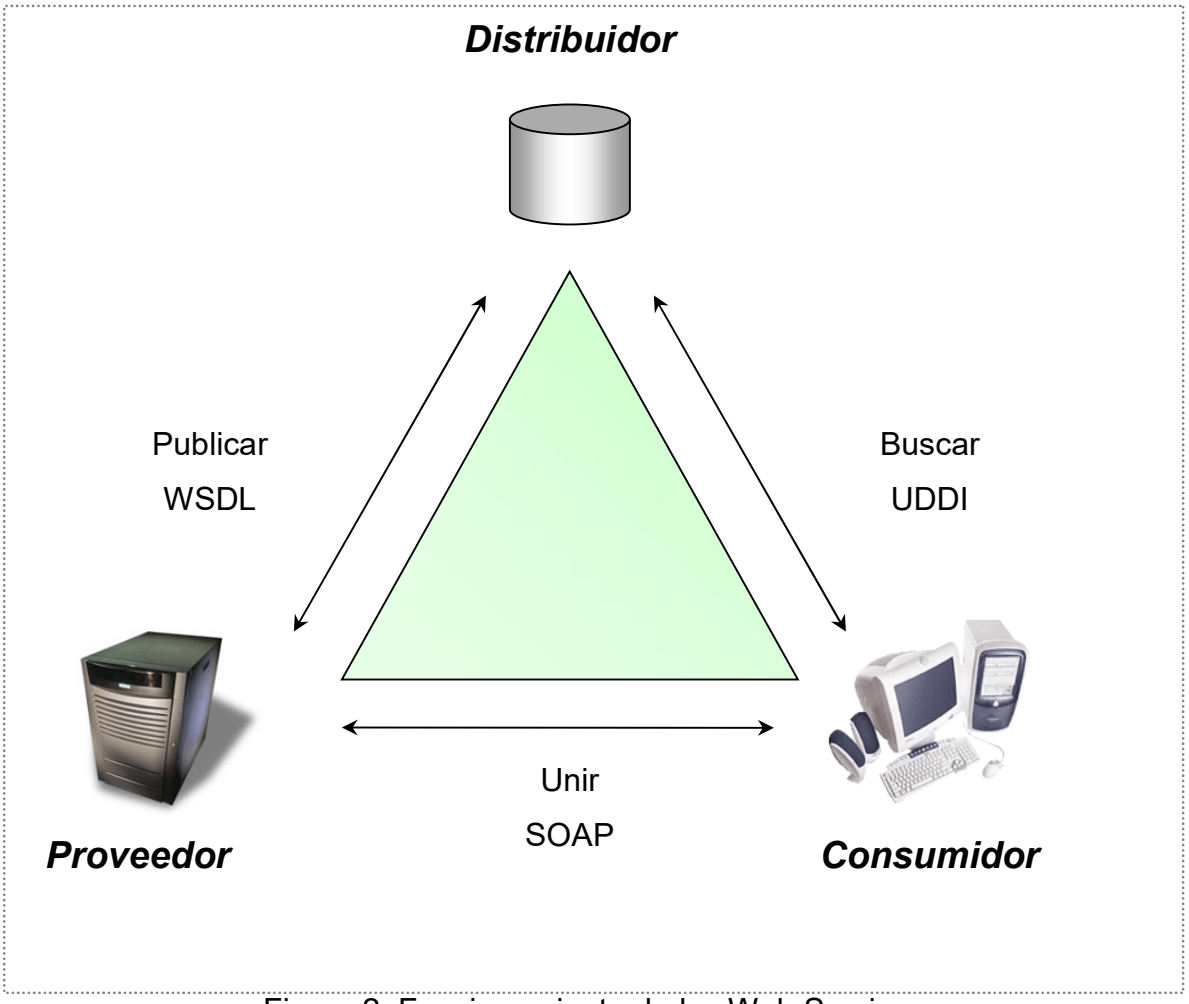



Figura 2. Funcionamiento de los Web Services

El proceso inicia cuando el **Proveedor**, quien es la entidad o persona que ha hecho el desarrollo, se dispone a **Publicar** su Web Service en el **Distribuidor**. Este proceso consiste en crear una cuenta en un **UDDI**, para publicar el archivo **WSDL**, así como para proporcionar información del Proveedor y clasificar cada Web Service según su taxonomía. Luego el **Consumidor**, o aplicación que consumirá algún servicio, se dispone a **Buscar** en el **UDDI** un servicio que se

acople a sus necesidades; cabe notar que la búsqueda la hará una persona que se encuentre en condiciones de comprender la sintaxis del archivo **WSDL**. Finalmente el **Consumidor** generará un cliente que consumirá el *Web Service* en base al archivo **WSDL** para que se pueda **UNIR** con el proveedor a través del uso de **SOAP** para envío y recepción de mensajes sobre HTTP.

## 2.5. PROTOCOLOS DE TRANSPORTE

Como los *Web Services* pasan sus mensajes sobre la Internet, estos mensajes pueden transitar sobre una variedad de protocolos de transporte, así que se mencionarán los que se podrían usar:

- HTTP (HyperText Transfer Protocol): Para transacciones web.
- SMTP (Simple Mail Transfer Protocol): Para el correo electrónico.
- FTP (File Transfer Protocol): Para transferencia de archivos.

Sin embargo, un número de críticos han argumentado que esos protocolos no fueron diseñados para soportar transacciones seguras así que se están desarrollando otros protocolos de transporte con soporte a los Web Services como:

- REST (REpresentational State Transfer)
- BEEP (Block Extensible Exchange Protocol)
- HTTP-R (Realiable HTTP)[9]

Con todo lo anterior, cabe notar que actualmente HTTP es la opción más popular para el transporte de servicios. HTTP es simple, estable, y altamente desplegado; fue definido por la IETF (Internet Engineering Task Force) en su RFC (Request For Comments) 2068 para sistemas de información de hipermedia, distribuidos y colaborativos. Consiste de consultas del cliente al servidor (request) y respuestas del servidor al cliente (response) en base a las conexiones TCP/IP. Entre sus métodos los más usados son GET y POST, para consultas pequeñas y grandes,

[9] CERAMI, Ethan. Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. Sebastopol: O'Reilly, 2002. P. 19.

respectivamente. Además la mayoría de los *firewalls* permiten el tráfico HTTP por lo tanto es el preferido para las RPC de los *Web Services*.

## 2.6. APORTE DEL CAPÍTULO PARA EL LECTOR

Los Web Services están diseñados para poder comunicar aplicaciones de manera transparente, ya que en vez de incluir código en el proveedor o en el cliente se agrega una capa aparte al esquema llamada distribuidor, la cual va a ser necesaria para iniciar la comunicación.

Este esquema de tres capas también lo maneja el middleware estilo RPC como CORBA, pero éstos usan tecnologías propietarias que no los hace tan accesibles e incrementa los costos de desarrollo. Es ahí donde los Web Services se diferencian al basarse sobre el protocolo estándar de transporte para Internet, como es HTTP y se complementa con protocolos estándares como WSDL/SOAP/UDDI que corresponden a lenguajes basados en el metalenguaje XML para brindar una completa interoperabilidad.

A la hora de su implementación existe una variedad de herramientas para hacerlo, y se pueden elegir de acuerdo a la necesidad del usuario o del mismo desarrollador. A continuación se nombrarán las que fueron evaluadas:

a) Apache Axis: Es un motor SOAP de Apache que está en capacidad de atender las solicitudes y respuestas entre dos aplicaciones. Se aloja como una aplicación dentro del Servidor Web Tomcat, también de la comunidad Apache, la cual luego de su despliegue se puede acceder en un navegador web con una URL conformada según la configuración así: http://dirección:puerto/webapp; donde dirección equivale a localhost, 127.0.0.1 o la dirección IP de un equipo local o remoto, puerto es el número asignado durante la instalación y webapp es el nombre de la subcarpeta desplegada dentro de la carpeta webapps, así que la URL más común es: http://localhost:8080/axis. En ella se pueden emplear clases nativas (*.java) o compiladas (*.class) para desplegarlas como Web Services. Recibe

instrucciones por consola para lo cual se hace necesario instalar el JDK (Java Development Kit), habilitar la función java, agregar un parser XML como Xerces y registrar los archivos jar en el Classpath. Por lo cual puede que a la hora de hacer el despliegue (deploy.wsdd) se encuentre un error respecto a una clase faltante en vez del mensaje de éxito.

b) Java Web Services Developer Pack: Desarrollado por Sun Microsystems para el empleo de las APIs que soportan a los Web Services. Requiere de una instalación previa de Apache Ant, herramientas que también pueden arrojar mensajes por clases faltantes al ejecutar sus comandos por consola, por lo cual no se pudo emplear ésta herramienta.

c) NetBeans IDE 5.0 con Sun Java System Application Server 8.2: Desarrollados por el proyecto NetBeans y Sun Microsystems, respectivamente. Requiere mínimo 512 MB de Memoria RAM y la instalación previa del JDK. El entorno proporciona un ejemplo de Web Service (HiWS) y un cliente que lo consume (HiWSClient).

Además de contar con una herramienta para el desarrollo de un Web Service también es necesario conocer la lógica del negocio para la cual se va a implementar ésta tecnología, ya que de lo contrario luego de ejecutar los ejemplos no se podría plantear un nuevo esquema de comunicación para un caso específico.

Cabe destacar que es necesario conocer las direcciones IP de los equipos involucrados en la comunicación y desactivar los firewalls (de software o hardware) para permitir el flujo de los paquetes SOAP que viajan sobre HTTP.

Finalmente se hace necesario aclarar que la tecnología “Web Services” se traduce del inglés al español como “Servicios Web”, pero en el contexto del presente trabajo de grado se mantuvo la palabra en inglés refiriéndose a la

tecnología que comunica aplicaciones a través de Internet pues en español se entiende servicios web como correo electrónico, contador de visitadores, etc.

### 2.6.1. ANTECEDENTES

La computación distribuida implica que varios equipos se deben interconectar entre sí para compartir información del proceso del negocio para el cual fueron diseñados, así como compartir recursos y encontrar nuevas maneras de generar ingresos y reducir costos en las organizaciones. Los diferentes mecanismos para lograrlo han ido evolucionando con el paso del tiempo, siempre buscando mejorar las falencias de sus precedentes. Los dos más populares trabajan con el esquema cliente/servidor: *sockets* y RPC.

Los *sockets* envían y reciben información a través de los puertos y las direcciones del protocolo TCP/IP que son asignadas, éstos pueden ser desarrollados en Java, Visual Basic o C++, entre otros lenguajes de programación; sin embargo dependen de la disponibilidad de los puertos y la conexión no puede ser tan flexible al estar dentro de una misma red. Las llamadas a procedimientos remotos o RPC son hechas desde un *stub* en el cliente que hace las llamadas a un *skeleton* en el servidor a través del protocolo IIOP y se basan en tecnologías propietarias como CORBA o RMI, entre otras; estas tecnologías resultan muy costosas y no son interoperables entre sí.

Luego se diseñó un nuevo mecanismo de conexión entre aplicaciones, los *web services,* que soportan la mensajería entre aplicaciones heterogéneas y viajan sobre el protocolo HTTP, como la WWW, el cual es aceptado por los *firewalls*, pueden ser desarrollados en plataformas como J2EE y Microsoft .NET las cuales contienen las librerías y APIs necesarias para su desarrollo, despliegue y registro sobre un servidor de aplicaciones y/o servidores web.

### 2.6.2. ESTADO DEL ARTE

Las soluciones actuales de middleware han sido consideradas una revolución, sin embargo como éstas no están creando nuevos paradigmas de programación entonces lo que en realidad son es una evolución que busca integrar los sistemas de información empresariales entre sí para aumentar la cadena de valor sin problemas de fronteras, gracias a las ventajas del Internet y XML.

#### 2.6.2.1. COMUNIDADES

A nivel internacional se destacan las siguientes empresas y organizaciones, que se han encargado de difundir estos nuevos temas a los desarrolladores para que lleguemos a un mismo futuro. Las comunidades se encuentran listadas alfabéticamente así:

- APACHE: Web Services Project <http://ws.apache.org/>
- APPLE: Developer Connection <http://developer.apple.com/internet/webservices/>
- DAML: Semantic Web Services <http://www.daml.org/services/>
- HP: Web Services Platform <http://www.hpmiddleware.com/HPISAPI.dll/hpmiddleware/products/hp_web_services/default.jsp>
- IBM: SOA and Web Services <http://www-136.ibm.com/developerworks/webservices/>
- IMAG: Adele Team – Project - SCORE: Service and Component-Oriented REsearch <http://www-adele.imag.fr/Research/components.html>
- Java Technology and Web Services <http://java.sun.com/webservices/index.jsp>
- Java Web Services and XML <http://community.java.net/java-ws-xml/>
- MICROSOFT: Web Services and Other Distributed Technologies Developer Center <http://msdn.microsoft.com/webservices/>
- ORACLE: Service-Oriented Architecture Technology Center <http://www.oracle.com/technology/tech/webservices/index.html>
- O'Reilly XML Semantic Web <http://www.xml.com/semweb/>

- QoS Labs: Service-Oriented <http://www.qoslabs.com/wb2/>
- REMOTE METHODS: Finding Web Service Providers <http://www.remotemethods.com/3>
- SEARCH WEB SERVICES: Information resource <http://searchwebservices.techtarget.com/>
- SEMANTIC WEB: Organization <http://www.semanticweb.org/>
- Semantic Web-Enabled Web Services http://www.hpl.hp.com/research/ssrc/competitive/web_services/
- SUN MICROSYSTEMS: Web Services <http://developers.sun.com/techtopics/webservices/>
- Web Services <http://webservices.xml.com/>
- WEB SERVICES ARCHITECT: Source of information <http://www.webservicesarchitect.com/>
- WEB SERVICES JOURNAL: Magazine <http://webservices.sys-con.com/>
- WEB SERVICES ORGANIZATION: Portal <http://www.webservices.org/>
- Web Services Product Directory <http://searchwebservices.bitpipe.com/data/tlist?pb=ka_swbsv_webservices>
- Web Services: Developing secure, effective applications <http://devresource.hp.com/drc/topics/web_services.jsp>
- X METHODS: Zinder http://www.xmethods.net/

2.6.2.2. CONFERENCIAS

Además de los portales mencionados anteriormente, una muestra de que existe un movimiento fuerte a nivel mundial que apoya la evolución de los web services se demuestra con las conferencias que se han hecho sobre estos temas, listadas alfabéticamente así:

- ApacheCon Europe: July 18-22, 2005 - Stuttgart, Germany <http://www.apachecon.com/2005/EU/index.html/e=MjAwNS9FVQ>
- ApacheCon US: December 10-14, 2005 -San Diego, California, USA <http://www.apachecon.com/2005/US/>

- Arquitectura de Web Services con Java J2EE y su impacto en el desarrollo de software y los negocios electrónicos: Septiembre 20 al 28, 2005 - Bogotá D.C., Colombia <http://www.acis.org.co/index.php?id=517>
- ESWC - European Semantic Web Conference: May 29 - June 1, 2005 - Heraklion, Crete, Greece <http://www.eswc2005.org/>
- GML Days - GML and Geo-Spatial Web Services Conference: July 18-22, 2005 - Vancouver, British Columbia, Canada <http://www.gmldays.com/>
- ICWS - IEEE International Conference on Web Services: July 11-15, 2005 - Orlando, Florida, USA <http://conferences.computer.org/icws/2005/>
- ICWS - IEEE International Conference on Web Services: July, 2006 - USA <http://conferences.computer.org/icws/2005/>
- ISWC - International Semantic Web Conference: November 6-10, 2005 - Galway, Ireland <http://iswc2005.semanticweb.org/>
- JOC - Java One Conference: May 15-18, 2006 - San Francisco, California, USA
- NWeSP - Next Generation Web Services Practices: August 22-26, 2005 - Seoul, Korea <http://www.nwesp.org/>
- STC - Semantic Technology Conference: March 6-9, 2006 - San Jose, California, USA <http://www.semantic-conference.com/>
- WS on WS - Web Services on Wall Street: Februay 1-2, 2005 - New York City, USA <http://lighthouse-partners.com/wsonws/home.htm>
- XXV Salón de Informática - Arquitecturas Empresariales de Software: Septiembre 29, 30, Octubre 1 - Bogotá, Colombia <http://www.acis.org.co/XXVsaloninformatica>

#### 2.6.2.3. EMPRESAS

Continuando con el listado de los grandes aportes de la academia, de la mano de este movimiento también viene la industria, así que vale la pena mencionar a aquellas empresas pioneras en su aplicación, entre otras:

- Afina Sistemas (España) <http://www.afina.es/>

- Amazon - AWS <http://www.amazon.com/gp/browse.html/102-6107760-9154515?%5Fencoding=UTF8&node=3435361>
- Aplicaciones Ltda. (Bogotá, Colombia) <http://www.aplicaciones.com.co/>
- Compucentro (Bogotá, Colombia) <http://www.compucentro.com.co>
- Confecámaras – RUE: Registro Único Empresarial <http://64.76.190.67/RUE_WebSite/default.aspx>
- Dayscript (Colombia) <http://www.dayscript.com/>
- Deloitte :: eMayor Project :: SMGOs <http://www.deloitte.com/dtt/section_node/0,1042,sid%253D28578,00.html>
- E-comercio (Medellín, Colombia) <http://www.e-comercio.com.co/>
- Ericsson - SMS y MMS (Colombia) <http://www.cintel.org.co/rctonline/noticia.php3?nt=3727&edicion=14>
- Google Web APIs <http://www.google.com/apis/>
- Magic Software – Magic Web services <http://www.magicsoftware.com/bin/en.jsp?enPage=InnerPage&enDisplay=view&enDispWhat=Object&enDispWho=tech_tools^l8&enZone=tech_t_co>
- MTBASE / SYBASE (Colombia) <http://www.mtbase.com/contenido/documento?id=4,00018>
- PSL - ACH (Medellín y Bogotá, Colombia) <http://www.psl.com.co>
- QoS Labs - Web Services Engine <http://www.qoslabs.com/wb2/QoS_Labs/QoS__Web_Services_Engine3>
- Software AG - ApplinX (España) <http://www1.softwareag.com/es/products/applinx/sol_examples/ws/default.asp>
- Suramericana de Seguros : PUBS (Bogotá, Colombia) <http://www.suramericana.com/home.aspx>
- The Globus Alliance :: OGSA (United Kingdom) <http://www.globus.org/ogsa/>
- Uniandes: Eleggua (Bogotá, Colombia) <http://sistemas.uniandes.edu.co/manager.php?id=867>
- Voz y Datos (Cali, Colombia) <http://www.vozydatos.us/>

- Web Studio e-solutions (Colombia, España y USA) <http://www.ebstudio.com/www2/webstudio/>
- Yahoo! Web Services (USA) <http://www.oreillynet.com/pub/a/network/2005/02/28/yahoo.html>

#### 2.6.2.4. ESTÁNDARES

Ahora aquellos que hacen una actividad más formal en torno al tema y se dedican a publicar estándares para la comunidad son:

- OASIS - UDDI <http://www.uddi.org/>
- OMG – MDA: Model Driven Architecture <http://www.omg.org/mda/>
- W3C – SOAP: Simple Object Access Protocol <http://www.w3.org/TR/soap/>
- W3C – Semantic Web <http://www.w3.org/2001/sw/>
- W3C – WSDL: Web Services Description Language <http://www.w3.org/TR/wsdl>
- W3C - Web Services Activity <http://www.w3.org/2002/ws/>
- WS-I – Web Service Interoperability <http://www.ws-i.org/>

#### 2.6.2.5. EVANGELISTAS

Finalmente cabe destacar a los personajes más destacados en este campo:

- JEFF BARR: Amazon Web Services <http://www.xml.com/pub/a/2004/03/31/barr.html>
- HUGO HAAS: W3C – Web Service Activity Lead <http://www.w3.org/People/Hugo/>
- TIM BERNES-LEE: W3C – Web Service and Semantic Web <http://www.adtmag.com/article.asp?id=7662>

# 3. J2EE APLICADO AL PROTOTIPO

Ésta arquitectura se encuentra basada en la edición estándar de Java y provee APIs para emplearlas a través de aplicaciones Java. Soporta diferentes transacciones como conexión con Bases de Datos, Servidores Web, Servidores de Aplicaciones y Web Services. La tecnología Java creada por Sun Microsystems ha diseñado tres plataformas de desarrollo para elegir según el dispositivo del cliente así:

- Java 2 Standard Edition: Para aplicaciones de escritorio.
- Java 2 Mobile Edition: Para dispositivos de mano como PDAs o celulares.
- Java 2 Enterprise Edition: Para aplicaciones distribuidas.

Existe varias versiones: 1.2, 1.3, 1.4 y 5. La última es llamada Java EE 5. En éste caso se empleó la especificación J2EE 1.4 lo cual permitió que varias empresas desarrollaran servidores de aplicaciones acogiéndose a sus lineamientos, y así Sun Microsystems los evaluó y afirma que en estos momentos 14 de los que existen en el mercado cumplen con ella.

Cabe anotar que cada Servidor de Aplicaciones sólo es compatible, la mayoría de las veces, con un IDE de la misma compañía y que una aplicación empresarial empaquetada o construida con una marca no puede ser desplegada en otra como consecuencia de la información de contexto almacenada en los archivos de configuración xml, properties y manifest.

En este caso se ha elegido J2EE, ya que se desarrollará un prototipo que emplea varios servidores (Base de Datos, Web y Aplicación). Consta de una colección de interfaces para desarrollar aplicaciones empresariales multi-capas basadas en componentes de Java.

### 3.1. API (Application Programming Interface) V1.4

Los componentes son piezas de software orientadas a la reutilización y según su estructura interna se pueden clasificar en:

- Caja blanca: Provee acceso al código fuente.
- Caja gris: Provee acceso a una API.
- Caja negra: Provee un ejecutable.

Las APIs o Interfaces de Programación de Aplicación son componentes que a través de una especificación dan a conocer qué métodos contienen y cuáles son los tipos de datos que usan como parámetros de entrada y salida.

La especificación J2EE provee un conjunto de especificaciones de APIs para que los desarrolladores empleen los diferentes productos de la tecnología Java en aplicaciones nuevas y portables. A continuación nombraremos las APIs que se usarán en el desarrollo del prototipo.

#### 3.1.1. JDBC (Java Database Connectivity)

Conectividad a Base de Datos, provee acceso entre un Sistema de Gestión de Base de Datos y un Servidor Web, una aplicación de escritorio, o un Servidor de Aplicaciones para hacer consultas desde clientes locales o remotos. Se puede interactuar con diferentes proveedores de servidores de base de datos y permite invocar comandos SQL desde métodos del lenguaje de programación Java. Para que funcione se debe crear la conexión en una clase y luego agregar un archivo **JAR** (Java ARchive) en la estructura física del servidor. Este archivo se puede descargar, teniendo en cuenta la versión de J2EE y el nombre del SGBD, del siguiente sitio en la web: http://developers.sun.com/product/jdbc/drivers

#### 3.1.2. JSP (JavaServer Pages)

Páginas del Servidor que contienen etiquetas HTML y Java, para el contenido estático y dinámico, respectivamente, que un cliente accede por medio de un navegador web. Necesitan de un motor compatible con JSP y Sevlets como:

- Apache Tomcat 5.5 <http://tomcat.apache.org/>

- Caucho Resin 3.0 <http://www.caucho.com/resin-3.0/>
- NewAtlanta ServletExec 5.0 <http://www.newatlanta.com/products/servletexec/index.jsp>

Es imperativo que las páginas con formato jsp se ubiquen dentro de una carpeta con el nombre de la aplicación, así como una subcarpeta llamada WEB-INF dentro de la cual se ubicará el archivo de configuración XML, denominado **web.xml**. De manera opcional pueden haber subcarpetas dentro de la carpeta WEB-INF con nombre **src** que contiene las clases java que se usarán como componentes o beans desde las páginas jsp y otra llamada **classes** con los archivos compilados de las clases java; entre otras definidas por el programador como una llamada img para ubicar las imágenes que usarán las páginas web. Luego esa carpeta se podrá empaquetar dentro de un **WAR** (Web application ARchive) para que pueda ser desplegada en algún Servidor Web con motor JSP/Servlets o de Aplicaciones y publicada en el navegador web, con una URL como la que se puede acceder por defecto: http://localhost:8080/web para un servidor con salida por el puerto 8080 y web nombre de la carpeta o archivo desplegado.

### 3.1.3. EJB (Enterprise JavaBeans)

Es un componente del lado del servidor que se encargan de encapsular la lógica del negocio y puede ser consumido por un Servlet o una página JSP. Según su interacción puede ser de alguno de los siguientes tipos:

- Mensaje: para el procesamiento de mensajes asíncronos.
- Entidad: para vincularlo directamente a una tabla de la base de datos.
- Sesión: para el procesamiento de mensajes síncronos para un cliente. Éstos de subdividen en dos clases, con estado (Statefull) y sin estado (Stateless).

Necesitan de un contenedor de EJBs, más conocidos como Servidor de Aplicaciones, los cuales deben ser compatibles con J2EE versión 1.4 como:

- Apache Jerónimo 1.0-M5 <http://geronimo.apache.org/>
- BEA WebLogic Server 9.0 <http://www.bea.com/diablo/index.html>
- CAS OnceAS2.0 Institute of Software, Chinese Academy of Sciences <http://www.once.com.cn/>

- IBM WebSphere Application Server <http://www-306.ibm.com/software/webservers/appserv/was/>
- JBoss Application Server 4.0 <http://www.jboss.org/>
- Kingdee Apusic v4.0 <http://www.apusic.com/product/index.htm>
- NEC WebOTX 6.1 <http://www.sw.nec.co.jp/middle/WebOTX-e/>
- ObjectWeb JOnAS v4.4 <http://jonas.objectweb.org/download/index.html>
- Oracle Application Server Containers for J2EE 10g (10.1.3) <http://www.oracle.com/technology/tech/java/oc4j/index.html>
- Sun Java System Application Server Platform Edition 8 <http://java.sun.com/javaee/index.jsp>
- Tmax Soft JEUS 5.0 <http://technet.tmaxsoft.com/en/index.do>
- TongTech TongWeb Application Server Version v4.6 <http://tongtech.com/english/product/index3.htm>
- Trifork T4 Application Server <http://www.trifork.com/products/T4/>[10]

Para su desarrollo también se hace necesario el empleo de un IDE (Integrated Development Environment) como Eclipse 3.1 o NetBeans 5.0, entre otros. Dentro de un paquete o carpeta se puede ubicar la clase Java que funcionará como EJB de sesión, mensaje o entidad. Luego en otro paquete se generarán las interfaces del EJB. Dentro de la carpeta del proyecto debe aparecer la subcarpeta META-INF, con el archivo **ejb-jar.xml** con descripción de los nombres internos de componentes y **sun-ejb-jar.xml,** para un paquete creado en Sun Java System Application Server, con información de los componentes (bean) empleados por el mismo.

### 3.1.4. JAX-RPC (Java API for XML-based RPC)

El API de Java para RPC basadas en XML se usa para construir aplicaciones web y web services usando estándares como SOAP y HTTP, así que los programas cliente pueden hacer llamadas a procedimientos remotos sobre Internet. JAX-RPC también soporta WSDL, así que se pueden importar y exportar documentos WSDL.

[10] SUN MICROSYSTEMS. Compatibility. [En línea] Disponible en Internet URL <http://java.sun.com/j2ee/compatibility.html>

## 3.2. CAPAS

Como se han venido nombrando aplicaciones multi-capas, a continuación se darán detalles sobre las capas (tiers) que se aprecian en una aplicación J2EE, así como lo demuestra la figura 3.


Figura 3. Niveles de una arquitectura multinivel en plataforma J2EE [11]

### 3.2.1. PERSISTENCIA

Hace alusión al servidor de Base de Datos. En nuestro se caso se emplea MySQL Server 5.0 junto con las aplicaciones:

- MySQL Query Browser 1.1: para la creación de tablas y relaciones.
- MySQL Administrator 1.1: para funciones de backup y restore.
- DBDesigner 4: para vincular el modelo relacional con la base de datos.

### 3.2.2. DOMINIO

Hace referencia al Servidor de Aplicaciones, en este caso se usa Sun Java System Application Server 8 el cual es compatible con J2EE 1.4 y por lo tanto contiene los recursos para consumir las APIs necesarias en el prototipo. Se

[11] FRANKY, María Consuelo. Evolución de Arquitecturas y Frameworks para sistemas J2EE. En: Salón de Informática. Arquitecturas Empresariales de Software: Espejismos y Realidades. (8°: 2005: Bogotá). Memorias del Salón de Informática. Bogotá, Asociación Colombiana de Ingenieros de Sistemas, 2005. Diapositiva 4.

administra desde el IDE NetBeans 5.0. Contiene de un archivo **application.xml** donde se declaran los módulos web y ejb que contenga el archivo, así como un archivo **sun-application.xml** con una especificación del servidor en el cual fue construido el EAR.

### 3.2.3 SERVICIOS

La capa de servicios se refiere a un ambiente externo de la aplicación, el cual colabora en una variedad de formas, en este caso son las APIS con soporte a los *Web Services* que vienen embebidas en el Servidor de Aplicaciones. Requiere de un archivo **ejb-jar.xml** con una etiqueta *service endpoint* donde se ubicará el paquete y el nombre de la clase que funciona como interfaz del bean que será expuesto como web service; así como un archivo **webservices.xml** que relaciona al EJB con su SEI (Service Endpoint Interface) y un archivo **mapping.xml**, por cada Web Service, donde definen los tipos de datos empleados como parámetros de entrada y de retorno de cada método expuesto mediante la interfaz.

### 3.2.4. APLICACIÓN

Se refiere a los componentes responsables de implementar la funcionalidad de una aplicación. Estos componentes deben manejar los datos de la aplicación y el estado mientras se ejecutan las operaciones específicas soportadas por la aplicación. Aquí se usa Apache Tomcat 5.5. Contiene un archivo llamado **web.xml** con la declaración del archivo principal, declaraciones de servlets y/o configuraciones de seguridad.

### 3.2.5. PRESENTACIÓN

Se refiere a aquellos componentes responsables de crear y administrar una interfaz de la aplicación con sus usuarios. Los navegadores web son los protagonistas acá, el más usado es Microsoft Internet Explorer 6.0 pero recomendamos el uso de Mozilla Firefox 1.5.

# 4. METODOLOGÍA RUP

El Proceso Unificado de Desarrollo de Software, RUP, tiene como su primer antecedente la Metodología Ericsson en la cual se introdujo el concepto de Casos de Uso; entre 1987 y 1995 sale al mercado el proceso de desarrollo Objectory; más adelante se lanza, entre 1995 y 1997, Racional Objectory Process (ROP), hasta que finalmente en 1998 sale al mercado Racional Unified Process 5.0 el cual ha ido cambiando hasta hoy en día.

El Proceso de Desarrollo Unificado es un proceso de la Ingeniería del Software, basado en componentes los cuales son interconectados a través de interfaces bien definidas. El objetivo principal del RUP es guiar a los desarrolladores de software en la implementación y distribución de sistemas ajustados a los requerimientos de los usuarios o clientes.

RUP (Rational Unified Process) junto con UML abarcan la metodología más usada para analizar, implementar y documentar sistemas orientados a objetos. Dentro de sus características se encuentra que es una metodología iterativa e incremental, dirigida por casos de uso y centrado en la arquitectura, para entenderlas un poco más éstas se describen a continuación:

- Iterativa e incremental: Para todo software complejo es indispensable dividir su proyecto en fases o ciclos de vida en los cuales se realizan varias iteraciones en donde se ejecutan varios trabajos o flujos para de esta manera obtener una revisión profunda de la funcionalidad del proyecto.
- Dirigida por casos de uso: Mediante los casos de uso los desarrolladores realizan modelos de diseño e implementación ayudando a la ejecución de

pruebas sobre los componentes que conforman al software; además éstos dirigen el desarrollo de la arquitectura.

- Centrado en la arquitectura: La arquitectura comprende los elementos más significativos del software la cual está influenciada por la plataforma, los sistemas operativos, las bases de datos, protocolos, sistemas heredados y los requerimientos no funcionales. A demás la arquitectura es realimentada por los casos de uso lo cual ayuda al desarrollo adecuado del software.

La arquitectura es necesaria para:

- Comprender el sistema.
- Organizar el desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.

El Proceso de Desarrollo Unificado se divide en cuatro fases y varias disciplinas, las cuales varían según la complejidad del proyecto, como se muestra en la figura 4.


Figura 4. Ciclo de vida del RUP[12]

[12] NEXTEL, Engineering. Soluciones a Medida: Metodologías de Desarrollo. [documento en línea] [Madrid, España] Disponible en Internet <URL: http://www.nexteleng.es/catalogos/a_medida.asp>

## 4.1. CONCEPCIÓN

Su objetivo principal es conocer los objetivos para el ciclo de vida del proyecto delimitando su alcance a través de la exploración del problema para luego decidir si se continúa el desarrollo del producto.

Los objetivos de esta fase son:

- Analizar el dominio del problema.
- Establecer la arquitectura.
- Desarrollar un plan de proyecto.
- Eliminar los elementos de mayor riesgo para el desarrollo del proyecto.

## 4.2. ANÁLISIS

Su principal objetivo es plantear la arquitectura para el proyecto, obteniendo los requerimientos funcionales y recolectando toda la información necesaria para iniciar con la fase de construcción del producto eliminando los posibles riesgos que puedan surgir durante el ciclo de vida del producto.

## 4.3. CONSTRUCCIÓN

El principal objetivo de esta fase es lograr la operación del producto haciendo iteraciones para obtener una versión Beta del producto para su operación y puesta en marcha. También es importante saber que en esta fase se deben tener los modelos completos del producto (Diagramas de Casos de Uso, de Clases, E/R y otros que requiera el usuario final).

## 4.4. TRANSICIÓN

El objetivo principal de esta fase es entregar el producto al usuario para su ejecución y posteriores cambios que sean necesarios en cuanto al software y a la documentación que se le dará al usuario para el manejo, ejecución e instalación del producto.

## 4.5. APORTES DEL CAPÍTULO PARA EL LECTOR

### 4.5.1. PRIMERA FASE

Para nuestro prototipo esta fase inició con la visita a dos arroceras de Santander, Arrocera La Aurora y Arroz San Cristóbal, en la primera se revisó, en general, el proceso de transformación del arroz ya que nuestro proyecto contempla el desarrollo de un prototipo del sistema de inventario por lo cual era importante conocer en un principio el proceso del arroz, y en la segunda arrocera se logró conocer el proceso de transformación del arroz de forma más detallada, incluso se realizaron varias visitas a las instalaciones de la arrocera para obtener la documentación del inventario que se lleva en ésta empresa. Luego con toda la información obtenida se crearon los primeros Diagramas de Casos de Uso del Negocio junto con sus respectivas especificaciones (Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 3. Arquitectura y Diseño > 3.3. Especificación de Requerimientos de Software del Módulo de Inventario).

### 4.5.2. SEGUNDA FASE

Luego de un estudio detallado del proceso del negocio se delimitó el alcance del proyecto llegando a la conclusión que lo primero que se iba a realizar era el prototipo del sistema de inventario para luego pasar a la elaboración de un Web Service que se encargue de comunicar, a través de protocolos, al sistema de contabilidad las cuentas por pagar, por cobrar y el estado de activos y pasivos de la empresa justo después de la compra de insumos o la venta de productos, como se puede apreciar en la figura 5.

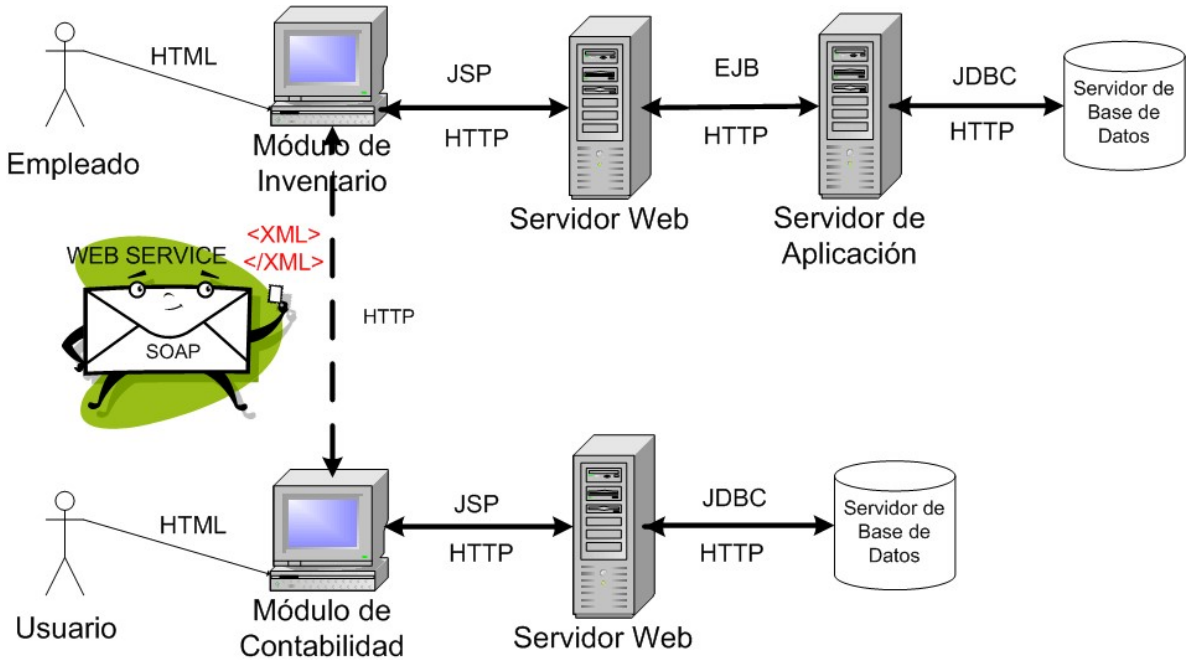



Figura 5. Arquitectura planteada del prototipo RISSO

En esta fase también se definió la arquitectura (J2EE) que iba a ser implementada en el proyecto, así como las herramientas a usar para el desarrollo del prototipo de inventario (Programación Orientada a Objetos, Páginas Dinámicas JSP y XML), el motor de Base de Datos (MySQL), Servidores (Tomcat y Sun Java System Application Server) y el IDE para el desarrollo del Web Services (Net Beans IDE 5.0). Cabe destacar que debido a la diversidad de herramientas existentes en el mercado se tornó difícil esta decisión para lo cual se debió, en algunos casos, probar algunas herramientas o servidores comparando su funcionalidad y rendimiento para luego optar por la opción más conveniente para el desarrollos del producto. Primero realizamos ensayos con Eclipse IDE 3.1 junto con JBoss 4.0.3SP1 para el desarrollo del Web Services, hicimos una prueba de una aplicación J2EE siguiendo el tutorial que trae consigo ésta herramienta, paso a seguir era montar un ejemplo de Web Services pero a medida que íbamos desarrollando empezaron a surgir inconvenientes como que el servidor JBoss 4.0.3SP1 no era el más adecuado para la ejecución de Web Services ya que ésta versión, a pesar de ser la más reciente, no daba soporte a los tags que se

requerían para la implementación del Web Services por lo cual, luego de investigar en la Internet, en un foro recomendaron trabajar con la versión JBoss 4.0.1SP1 ya que ésta si daba soporte a los tags necesarios para nuestro desarrollo; sin embargo después de consultas nos dimos cuenta que éste IDE no era lo suficientemente eficiente para trabajar el desarrollo de Web Services ya que al momento de generar los archivos propios para el despliegue de un Web Service (webservices.xml, ejb-jar.xml, entre otros) esto archivos tenían que ser reconfigurados a mano por lo cual la tarea se tornaría aún más larga y dispendiosa. Por último, y al final de tantas búsquedas, encontramos una herramienta que proporciona eficiencia, funcionalidad y rendimiento ya que nos proporciona el desarrollo de los Web Services de una manera más fácil y rápida; Net Beans IDE 5.0, el cual puede trabajar con servidores como Tomcat. JBoss y Sun Java System Application Server, en nuestro caso usamos ésta IDE con Sun Java System Application Server 8 ya que por experiencia sabemos que éste servidor junto con ésa versión son las mejores herramientas para el soporte de Web Services y que JBoss no daría el soporte necesario para su desarrollo.

### 4.5.3. TERCERA FASE

El proyecto lo conforma por una parte el prototipo de inventario el cual son páginas jsp desarrolladas con Java, HTML y JavaScript; éste prototipo se empleará para llevar registros en tiempo real de todo el proceso que conlleva la elaboración del arroz en la trilladora, por otra parte se encuentra el Módulo de Contabilidad en el cual mediante Web Service se comunicará con el módulo de inventario para asignar las cuentas por pagar, por cobrar y el estado de activos y pasivos a la base de datos de contabilidad así como en la figura 6.

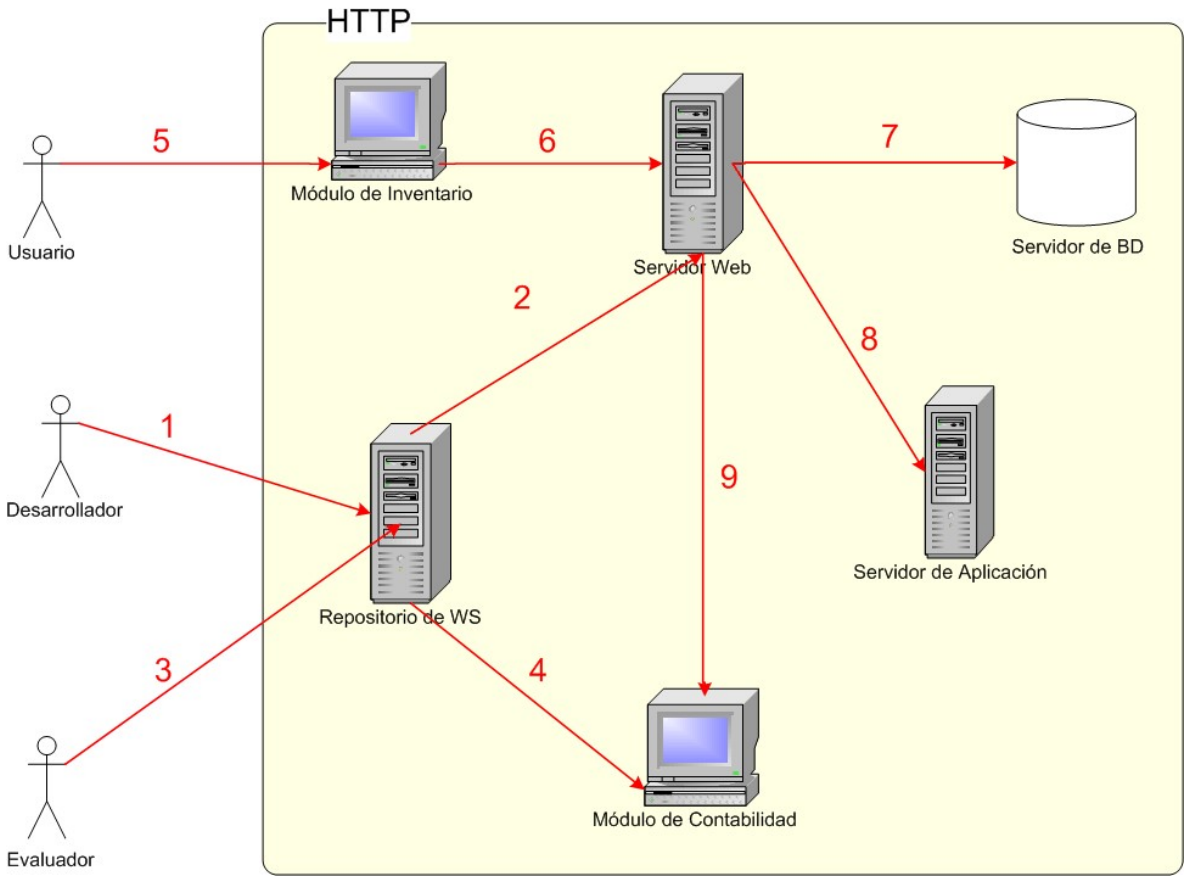



Figura 6. Funcionamiento de los Web Services en el prototipo RISSO

El esquema de la figura representa la interacción de los usuarios con el prototipo, así como la interacción entre las aplicaciones y los servidores sobre el protocolo de transporte estándar de Internet HTTP. Es un híbrido entre las figuras 2 y 3 de este documento, pero aplicadas a este caso real comprendiendo los siguientes pasos:

1. El Desarrollador del Prototipo RISSO se dispone a crear una cuenta en el Distribuidor, proporcionando información del servicio y de la empresa.
2. El Distribuidor (Repositorio UDDI) confirma la existencia de un archivo WSDL en la URL proporcionada por el Desarrollador para culminar el proceso de publicación.
3. El Evaluador se encarga de buscar en el Distribuidor algún Web Service que se acomode a las necesidades de la empresa, en este caso para comunicarse a través de la web con la Aplicación de Inventario.

4. El Consumidor (Módulo de Contabilidad) genera un cliente capaz de consumir al web service en base al archivo WSDL para recibir los mensajes correspondientes a cuentas por pagar, por cobrar y al estado de activos y pasivos sincrónicamente.
5. El Usuario accede al Módulo de Inventario, en un explorador web como Mozilla Firefox 1.5, y procede a hacer clic sobre una de las acciones que generará comunicación entre aplicaciones, Generar liquidación para cuentas por cobrar o Agregar factura para cuentas por pagar.
6. La petición HTML pasa al Servidor Web, para este caso Apache Tomcat 5.5, el cual hace las veces de nivel de Aplicación por contener los JSP y Proveedor del Web Service por contener el archivo WSDL.
7. La solicitud llega a la Base de Datos a través de JDBC para hacer consultas SQL. El nombre Persistencia hace alusión al nivel de almacenamiento de la arquitectura del prototipo para la cual se usó MySQL Server 5.0.
8. Si la consulta de agregar un registro es exitosa y devuelve un valor, se pasan los datos que necesita el Bean de Sesión Sin Estado para enviarlos como parámetros de entrada en el Web Service. Este EJB reside en un Servidor de Aplicaciones para este caso se eligió Sun Java System Application Server 8, el cual recibe el nombre de Dominio por contener el EJB y Servicios por contener la interfaz del Web Service.
9. El mensaje es empaquetado dentro de un sobre SOAP que será transportado sobre la Internet hacia el Consumidor.

También se encuentra un completo desarrollo de los Diagramas de Casos de Uso, de Clases, de Secuencia y el modelo Relacional del Módulo de Inventario (Ver anexo 7.2), así como las especificaciones funcionales del software del Módulo de Inventario y Contabilidad realizadas de acuerdo al lenguaje UML y a la metodología RUP (Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 3. Arquitectura y Diseño), así como las pruebas del prototipo las cuales se encuentran detallan en el anexo 7.3 al final de este documento.

### 4.5.4. CUARTA FASE

Los resultados de esta fase se pueden encuentran en la carpeta de las plantillas de la siguiente manera:

1. Manual de Usuario. (Ver CD > Carpeta Manuales > Archivo Manual_Usuario)
2. Manual Técnico. (Ver CD > Carpeta Manuales > Archivo Manual_Técnico)
3. Diagrama de Componentes del Módulo de Inventario. (Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 5. Entrega e Instalación > 5.4. Diagrama de Componentes del Módulo de Inventario).
4. Diagrama de Componentes del Módulo de Contabilidad. (Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 5. Entrega e Instalación > 5.5. Diagrama de Componentes del Módulo de Contabilidad).
5. Diagrama de Componentes del Módulo de Conexiones. (Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 5. Entrega e Instalación > 5.6. Diagrama de Componentes del Módulo de Conexiones).
6. Diagrama de Componentes del Módulo de Cliente. (Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 5. Entrega e Instalación > 5.7. Diagrama de Componentes del Módulo de Cliente).
7. Diagrama de Componentes del Prototipo RISSO. (Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 5. Entrega e Instalación > 5.8. Diagrama de Componentes del Módulo del Prototipo RISSO).
8. Diagrama de Despliegue del Prototipo RISSO. (Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 5. Entrega e Instalación > 5.9. Diagrama de Despliegue del Prototipo RISSO).

Luego de desplegados los paquetes en los servidores que conforman las diferentes capas del prototipo, se forma una Arquitectura Orientada a Servicios

donde el Módulo de Inventario es el Proveedor, el Módulo de Contabilidad es el Consumidor y el Distribuidor puede ser un repositorio público o local, como se aprecia en la figura 7.

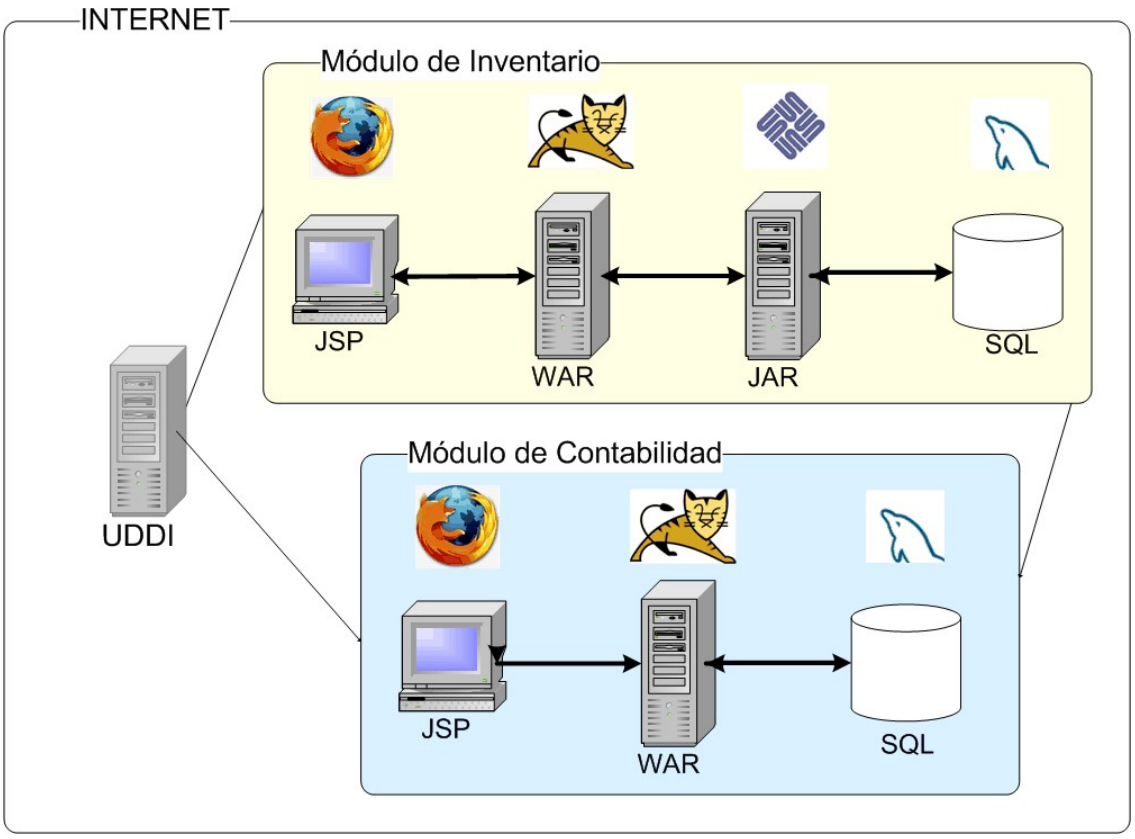



Figura 7. Arquitectura real del prototipo RISSO

# 5. CONCLUSIONES Y RECOMENDACIONES

- La metodología RUP plantea una serie de roles como: Analista, Diseñador, Especificador, Arquitecto, Integrador, Implementador, Probador y Administrador, donde cada uno debe ser asumido por una sola persona en el mejor de los casos, sin embargo en la realidad ésta asignación depende de la cantidad de personas involucradas en el proyecto lo cual afecta el tiempo de cada iteración en el proceso de desarrollo.

- Los diagramas UML son una gran ayuda como resultados de la fase intermedia, entre la concepción y la construcción según RUP, pero se hacen más comprensibles si se encuentran acompañados de sus respectivas especificaciones.

- Efectivamente la abstracción que se hace con los diagramas UML permiten hacer una visualización previa de las funciones y contenidos que puede llegar a tener la aplicación luego del desarrollo.

- Las especificaciones son documentos claves para el mantenimiento de una aplicación y para esto existen unas plantillas con lineamientos específicos para realizarlas, pero las más comunes vienen en formato .doc y tienen información redundante o pueden ser adquiridas a precios muy elevados; por lo tanto se eligieron las plantillas llamadas ReadySET desarrolladas por Tigris las cuales vienen en formato XHTML con CSS y están libres de licencias, lo cual las hace una opción interesante para cualquier proyecto de desarrollo de software.

- Para diseñar los diagramas de casos de uso, de clases y de secuencia inicialmente se eligió la herramienta Enterprise Architect 4.5, de la cual se descargó un trial con algunas limitaciones como deshabilitar su ejecución después de 30 días de uso y la inclusión de un letrero en marca de agua en la impresión de los diagramas; así que se optó por usar Rational Rose Enterprise Edition 2003, bajo licencia educativa, la cual permite interacción de los elementos en diferentes diagramas UML.

- El Sistema de Gestión de Base de Datos elegido inicialmente fue PostgreSQL 8.1 con pgAdmin III, el cual funciona bien para servidores remotos pero de manera local es difícil su configuración. Así que optamos por usar MySQL Server 5.0, ya que está libre de licencia y brinda gran soporte con sus herramientas de acceso y administración como Query Browser 1.1 y Administrator 1.1, así como la gran cantidad de drivers o controladores para diferentes plataformas.

- Para la elaboración del Diagrama Relacional de la Base de Datos se optó por usar la herramienta Power Designer 11, la cual solicitaba una licencia o de lo contrario caducaba a los 30 días; luego se eligió DB Designer 4 la cual es especial para MySQL, hace Ingeniería Inversa, es amigable al usuario y tiene licencia *freeware*.

- Apache Tomcat es un contenedor de Servlet y JSP, del cual se implementó inicialmente la versión 4.0, con la cual resultaba tedioso el proceso de despliegue de aplicaciones al tener que ejecutar un archivo por lotes (formato .bat) para iniciar y parar el servidor; lo cual fue mejorado al implementar la versión 5.5 mediante un monitor que se sitúa como icono en la barra de tareas para un mejor acceso.

- Para la implementación de web services fue necesario probar varias herramientas entre:
  - JBoss 4.0.3SP1 con Eclipse 3.1 y el plugin JBossIDE 1.5, ésta versión de servidor de aplicaciones tiene un error en el llamado a EJB desde un cliente por medio del método *lookup*.
  - JBoss 4.0.1SP1 con Eclipse 3.1 y el plugin JBossIDE 1.5, ésta versión de servidor de aplicaciones no tiene soporte para Web Services al basarse en XDoclet 1.2.3.
  - NetBeans IDE 5.0 con JBoss 4.0.3SP1, el IDE sólo es compatible con ésta versión del Servidor pero de antemano se conocía su error (bug).
  - NetBeans IDE 5.0 con JBoss 4.0.1SP1, el IDE no es compatible con ésta versión de Servidor.
  - NetBeans IDE 5.0 con Sun Java System Application Server 8.2, funcionó correctamente para el desarrollo y el despliegue pero exige mínimo 512 KB de Memoria RAM.

- El cliente puede acceder la aplicación desde un explorador de Internet como Internet Explorer 6.0, sin embargo éste guarda en memoria caché las páginas de la aplicación y no reconoce los cambios recientes; por esto se recomienda el uso de Mozilla Firefox 1.5, en el cual se evita éste inconveniente.

- Con respecto a la arquitectura multicapa, en el mejor de los casos cada capa lógica ocupa una capa física; pero en nuestro caso MySQL Server y Apache Tomcat pueden residir en un mismo equipo, pero Apache Tomcat y Sun Java System Application Server no lo hacen incluso saliendo por diferentes puertos se bloquean los clientes web.

- Para la exposición de EJBs de sesión sin estado como Web Services se hace necesario comprender a fondo tanto el funcionamiento de una herramienta de desarrollo como la tecnología a implementar.

- A la hora de elegir tecnologías y herramientas hay que tener en cuenta que la prioridad debe ser cubrir con las necesidades requeridas por el usuario en lugar de experimentar con lo más reciente.

- Generalmente los sistemas de información empresariales han sido diseñados para cubrir una necesidad específica, pero ahora el enfoque está en diseñar aplicaciones integrales o integrar las aplicaciones que ya existen y cumplen esas tareas por separado. Es ahí donde los Web Services tienen un gran mercado, el cual seguirá en expansión con el auge de las aplicaciones web que conllevarán a la formación de la web semántica.

- La conexión entre las aplicaciones se pudo haber realizado con otra consulta más de **JDBC**, ya que los módulos a conectar están desarrollados con JSP. Sin embargo para aplicaciones heterogéneas no se habría podido hacer así, sino con middleware como sockets (con TCP/IP) o RPC (con tecnologías propietarias), así que los Web Services son la evolución ya que funcionan sobre el protocolo más empleado de Internet, HTTP.

- La especificación J2EE empaqueta aplicaciones conformadas por componentes, pero cómo éstos han sido diseñados para una lógica de negocio en particular, **no son completamente reutilizables**. Esto se confirma con la descontinuación de los repositorios públicos, como el de Microsoft y el de IBM.

- Se lograron superar los objetivos planteados en el trabajo de grado precedente a éste sobre la misma temática, "*Creación y aplicación de una guía para el desarrollo de web services usando J2EE*", quienes en realidad hicieron su desarrollo usando la herramienta **AXIS**.

- Las páginas dinámicas que soportan las consultas a la Base de Datos se desarrollaron con lenguaje JSP (Java Server Pages), el cual fue necesario complementar con código JavaScript para las validaciones de campos

requeridos y/o numéricos. Por lo tanto se recomienda para una futura versión de éste prototipo usar un framework para aplicaciones web que facilite el desarrollo con etiquetas predefinidas como: JSF (Java Server Faces), Struts o AJAX.

- Los componentes EJB que fueron expuestos como Web Services en este prototipo reciben uno o dos parámetros y retornan un parámetro o un componente, entre los cuatro que fueron desarrollados. Se recomienda reemplazar el componente por un vector para comparar su rendimiento con el componente, y también reemplazarlo por un documento XML para evitar el mapeo de los tipos de datos java a tipos complejos XML y asignar los datos a una NXD (Native XML Database) donde recomendamos usar la herramienta OpenSource OpenLink Universo.

- Para probar la característica de la interoperabilidad que brinda el empleo de Web Services como esquema de comunicación entre dos aplicaciones web se recomienda desarrollar un cliente web en otro lenguaje de programación diferente al del proveedor.

## 6. REFERENCIAS BIBLIOGRÁFICAS


- BELLAS, Fernando et al. Tema 1: Introducción a J2EE. [en línea]. Integración de Sistemas, Dpto. de Tecnologías de la Información y las Comunicaciones, Universidade da Coruña. Fecha de publicación: Noviembre 21 de 2002. Disponible en Internet <URL: http://www.tic.udc.es/~fbellas/teaching/is-2002-2003/Tema1.pdf>

- CERAMI, Ethan. Web Service Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. First Edition. O'Reilly & Associates: Sebastopol. 2002.

- DAVIS, Scott et al. JBoss at Work: A Practical Guide. O'Reilly Media, Inc: Sebastopol. 2006.

- GABRICK, Kurt A. WEISS, David R. Java 2EE and XML Development. Manning Publications Co: Greenwich. 2002.

- GEROIMENKO, Vladimir. CHEN, Chamei. Visualizing the Semantic Web: XML-based Internet and Information Visulization. Springer: Singapore. 2003.

- JACOBSON, Ivar et al. El Lenguaje Unificado de Modelado. Madrid: Addison Wesley Iberoamericana, 1999. p. 11-12.

- [--------]. El Proceso Unificado de Desarrollo de Software. Madrid: Pearson Educación S.A., 2000. p. 10-12.

- MANTILLA ARIAS, Marlon y CAÑIZALES, Javier Leonardo. Creación y aplicación de una guía para el desarrollo de Web Services usando J2EE. Bucaramanga, 2004. 100 p. Trabajo de Grado (Ingeniero de Sistemas). Universidad Autónoma de Bucaramanga. Facultad de Ingeniería de Sistemas.

- PANDA, Debu. Turn EJB components into Web services: Build and deploy an EJB component as a Web service with JAX-RPC. Java World. Fecha de Publicación: Agosto 2 de 2004. [Artículo]. Disponible en Internet <http://www.javaworld.com/javaworld/jw-08-2004/jw-0802-ejbws.html>

- SINGH, Inderjeet et al. Designing Enterprise Applications with J2EE Platform., Second Edition. Sun Microsystems: California. 2002.

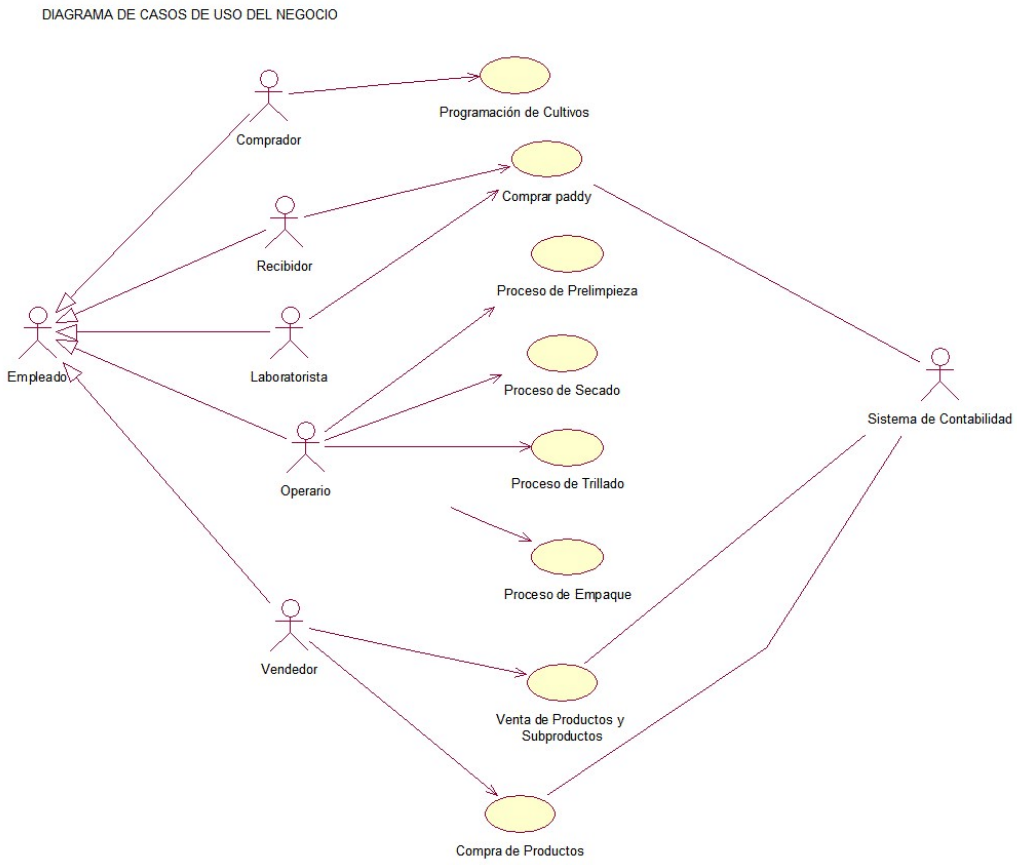
- THOMAS MANES, Anne. Web Services: A manager's guide. Boston: Addison-Wesley, 2003. 352 p.

- TIGRIS. [en línea]. ReadySet. Fecha de Publicación: Julio 8 de 2004. Disponible en Internet <URL: http://readyset.tigris.org/es/>

# 7. ANEXOS

## 7.1. RESULTADOS DE LA FASE DE CONCEPCIÓN

### 7.1.1. Diagrama de Casos de Uso del Negocio




### 7.1.2. Especificaciones de Casos de Uso del Negocio

(Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 2. Requerimientos y Especificaciones).

7.1.3. Glosario de Términos del Negocio

- **Administrador:** Este usuario está encargado de hacer la gestión de administración, el cual incluye acciones relacionadas con Usuario, Presentación y Tarifas.
- **Almacenaje:** Proceso que implica el reposo temporal del Arroz en bodegas entre diferentes procesos.
- **Arroz Blanco Entero:** Se obtiene a través de un proceso denominado blanqueo. Una vez descascarillado, se deben eliminar todas las capas que envuelven al grano de arroz, lo cual también se lleva a cabo mediante máquinas. El resultado es lo que conocemos como arroz blanco, que esta desprovisto de salvado y de germen. Este tipo de arroz, desafortunadamente, resulta más apetecible para la mayoría de la gente aunque no posee la calidad nutricional del integral ya que la mayoría de los nutrientes se encontraban en la cáscara.
- **Arroz Integral: E**s aquel arroz que no se le ha sometido al proceso de blanqueo, es decir que no se le ha quitado la cubierta externa o pericarpo del grano. Por lo tanto, presenta un color más oscuro consecuencia de los minerales que esta posee. Es un arroz que requiere una cocción más larga aunque con una calidad alimentaria superior dado que la mayoría de los nutrientes esenciales se encuentran en esta capa del grano que se elimina al someterlo a blanqueo.
- **Bodega:** Lugar donde es almacenado el arroz de manera temporal.
- **Cabezote:** También llamado arroz partido grande o Cristal, cuyo porcentaje de grano partido se encuentra entre 50% y 75%, y se vende como insumo para la fabricación de pastas alimenticias, sopas y cervezas.
- **Cárcamo:** Rejilla donde se descarga el Paddy verde húmedo que llega al molino.

- **Cicloventadora:** Máquina que recibe el grano seco el cual pasará a través de unos cilindros descascaradores, que removerán la cascarilla del grano, para así obtener como producto el arroz y como subproducto el tamo.
- **Comprador:** Este usuario está encargado de hacer la gestión de compras, el cual incluye acciones relacionadas con Finca, Cosecha, Proveedor e Insumo.
- **Cosecha:** Arroz cultivado en una finca por un cultivador y se divide en diferentes entregas.
- **Criba:** Cada uno de los aparatos mecánicos que se emplean en agricultura para cribar semillas, o en minería para lavar y limpiar los minerales.
- **Cultivador:** Es la persona propietaria de una finca.
- **Descascaradota:** Máquina que se encarga de retirar la cáscara al arroz Paddy seco, obteniendo los dos primeros subproductos del proceso: el arroz integral (cargo) o brown y la cascarilla de arroz.
- **Empaque:** Proceso mediante el cual los diferentes subproductos son almacenados en bolsas de diferentes presentaciones y peso según el mercado objetivo.
- **Empaquetado:** Proceso mediante el cual el Arroz Blanco, el Arroz Integral y el Arroz Partido son depositados en empaques de diferentes presentaciones y peso según el mercado objetivo.
- **Granza:** También llamado arroz partido pequeño, siendo todo pedazo de arroz tal que su longitud sea inferior a ¼ del tamaño normal del grano; y se utiliza en la preparación de concentrados para animales y cerveza.
- **Harina o Salvado de Arroz:** Subproducto obtenido en el proceso del pulido para la obtención de arroz blanco para consumo humano. Está constituido por parte de la almendra harinosa, la capa de aleurona y el germen, y representa del orden del 8% del peso del grano.
- **Humedad:** Es el porcentaje relativo a una muestra del grano recibido y relacionado con el nivel de agua contenido en el grano.

- **Impureza:** Es el porcentaje relativo al peso relacionado con los elementos que puedan acompañar al paddy después de su recolección ajenos al proceso de producción como ramas, hojas, entre otros.
- **Laboratorista:** Este usuario está encargado de hacer la gestión de laboratorio, el cual incluye acciones relacionadas con las pruebas de humedad e impureza que se le hace en el laratorio al paddy recibido.
- **Liquidar:** Generar una cuenta por pagar en el subsistema de contabilidad por concepto de la compra de insumo para la producción de arroz.
- **Molino:** Máquina para moler, compuesta de una muela, una solera y los mecanismos necesarios para transmitir y regularizar el movimiento producido por una fuerza motriz, como el agua, el viento, el vapor u otro agente mecánico.
- **Operario:** Este usuario está encargado de hacer la gestión de procesos, el cual incluye acciones relacionadas con los procesos de prelimpieza, secado, trillado y empaque; asi mismo cada uno de ellos está relacionado con Bodega, Producción y Subproductos.
- **Paddy:** También conocido como Paddy verde. Arroz que ha sido recolectado o aún está en el campo. También se denomina así al campo especialmente irrigado o inundado donde se produce el arroz.
- **Prelimpieza:** Proceso que reduce el porcentaje de impureza con el cual se recibe el Paddy.
- **Proceso:** Secuencia de acontecimientos definida única y delimitada, que obedece a una intención operacional en condiciones predeterminadas.
- **Pulimento:** Proceso en el que se le da el acabado final al Arroz.
- **Recibidor:** Este usuario está encargado de hacer la gestión de entregas, el cual incluye acciones relacionadas tan sólo con las entregas.
- **Repila:** Subproducto obtenido después del proceso de pulimento su principal característica es que su tamaño es ½ de arroz blanco.
- **Secado:** Proceso que implica suministrar calor a los silos para que el grano disminuya su porcentaje de humedad.

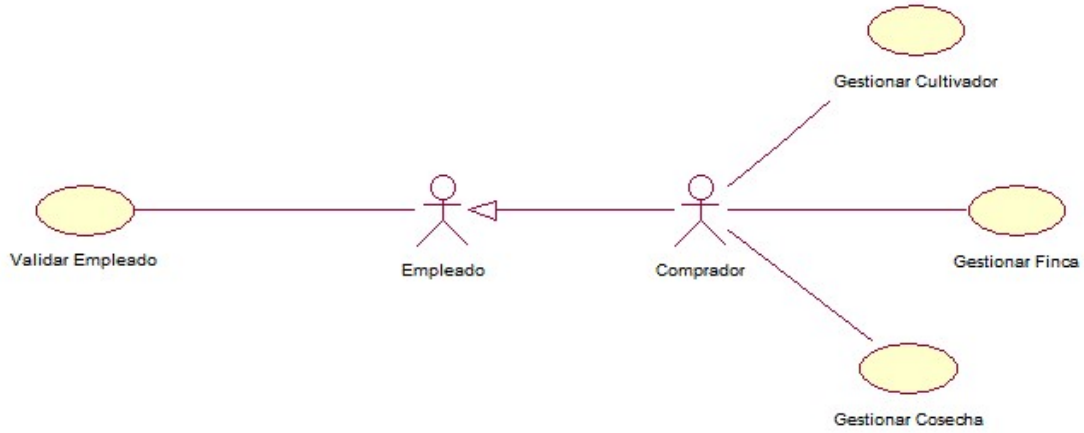
- **Separadora:** Máquina que se encarga de la separación de grano entero y partido, la cual posee zarandas de movimiento circular, llamadas rotativas o de roto vaivén, cuya acción enérgica asegura que el grano se distribuya en toda la superficie de criba disponible.
- **Silo:** Compartimiento en donde es depositado el PADDY para recibir transmisiones de calor que le permitan reducir su porcentaje de humedad.
- **Tamo:** Paja o cáscara obtenida luego del proceso de trillado.
- **Tamera:** Sitio en el cual es almacenado el Tamo o Cáscara de Arroz.
- **Trillado:** Proceso que implica remover la cáscara que envuelve al Arroz para continuar con el proceso de producción.
- **Vendedor:** Este usuario está encargado de hacer la gestión de ventas, el cual incluye acciones relacionadas con Cliente y Factura.
- **VDA:** Máquina pulidora de arroz blanco la cual le otorga un brillo estándar al mismo.
- **VDF:** Máquina pulidora de arroz blanco la cual le otorga un brillo óptimo al arroz para garantizar una mejor calidad del mismo.

## 7.2. RESULTADOS DE LA FASE DE ANÁLISIS

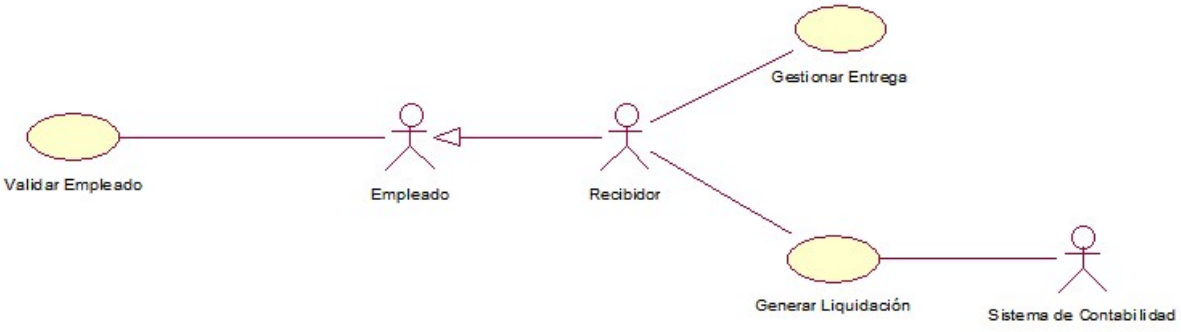
### 7.2.1. Diagrama de Casos de Uso del Módulo de Inventario

#### 7.2.1.1. Actor Empleado


#### 7.2.1.2. Actor Comprador

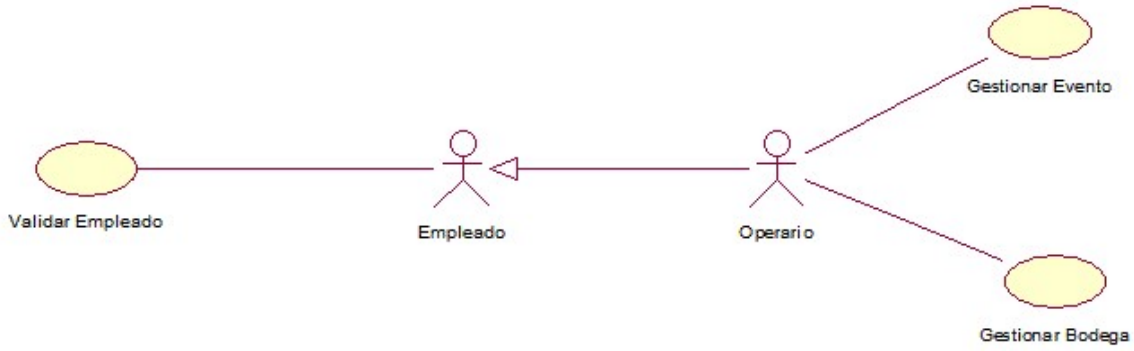



#### 7.2.1.3. Actor Recibidor

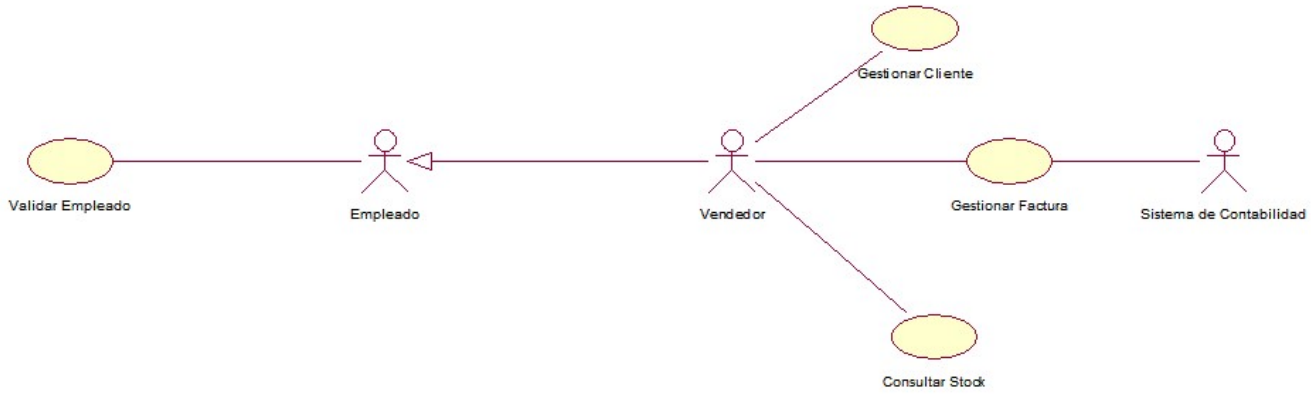


### 7.2.1.4. Actor Laboratorista

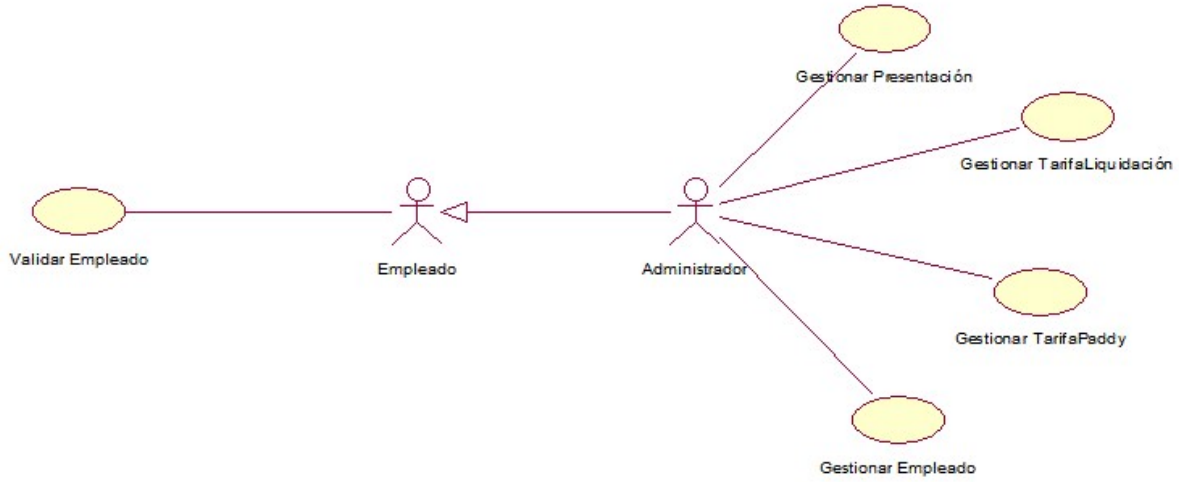

### 7.2.1.5. Actor Operario




### 7.2.1.6. Actor Vendedor



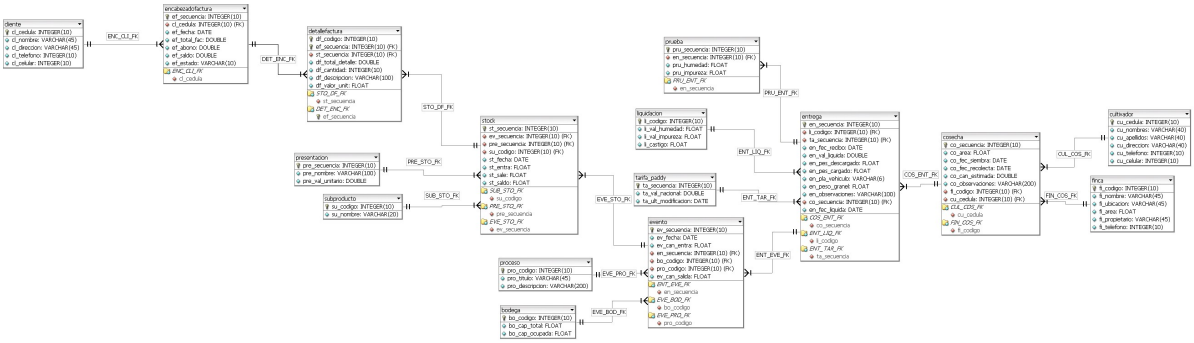
### 7.2.1.7. Actor Administrador



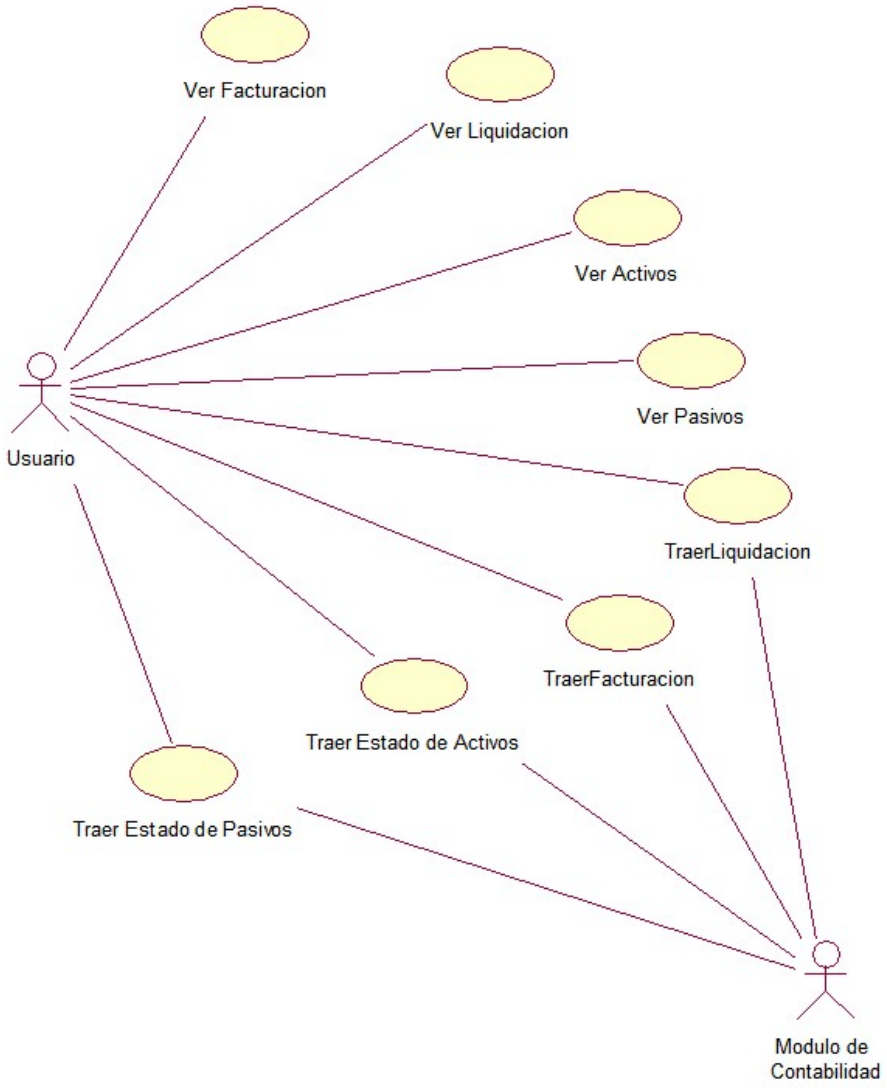

### 7.2.2. Especificación de Casos de Uso del Software del Módulo Inventario

(Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 3. Arquitectura y Diseño > 3.3. Especificación de Requerimientos de Software del Módulo de Inventario).

### 7.2.3. Diagrama Relacional de la Base de Datos de Inventario

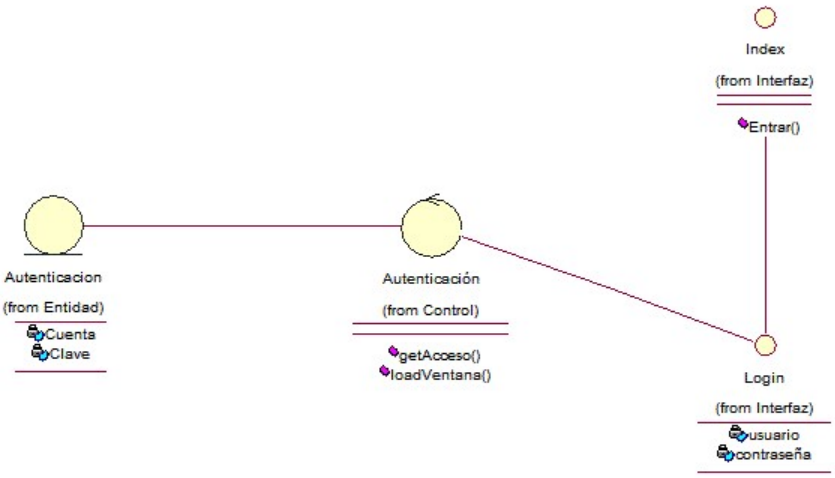


7.2.4. Diagrama de Casos de Uso del Módulo Contabilidad

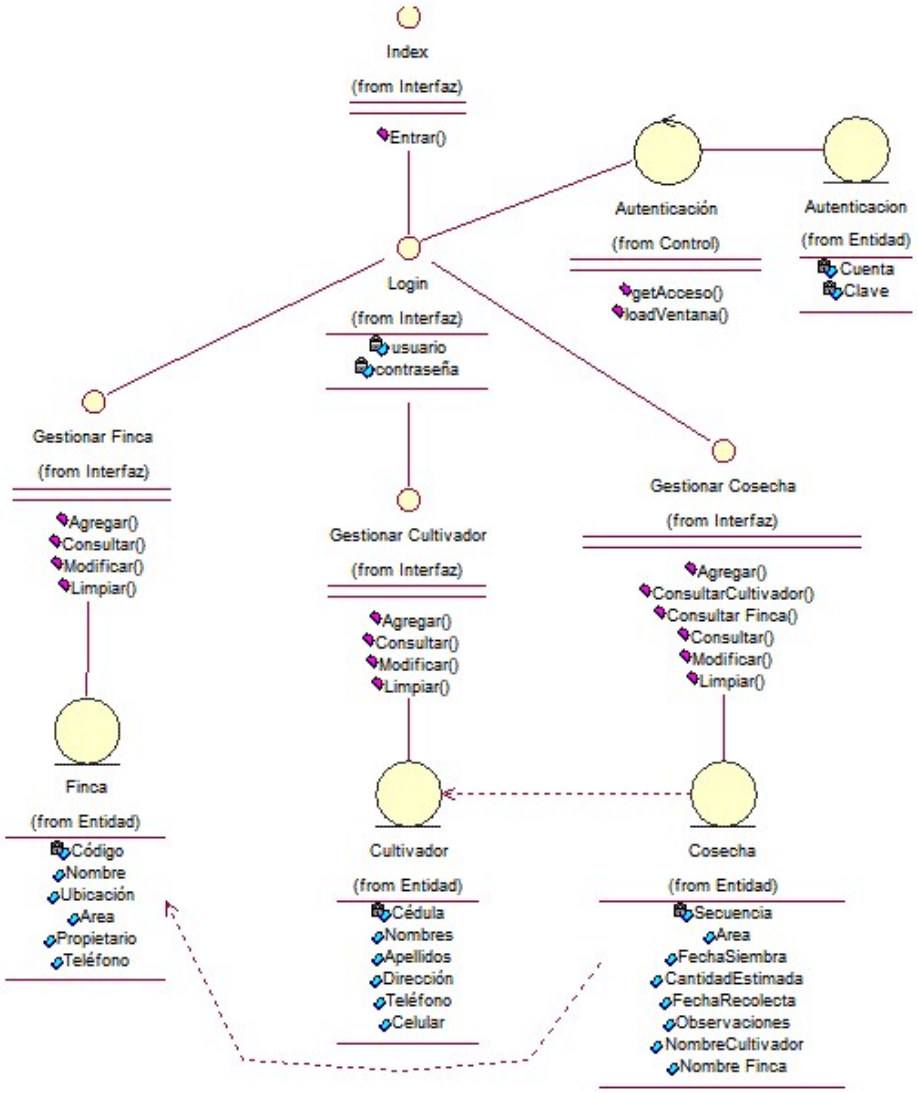


### 7.2.5. Diagrama de Clases del Módulo de Inventario

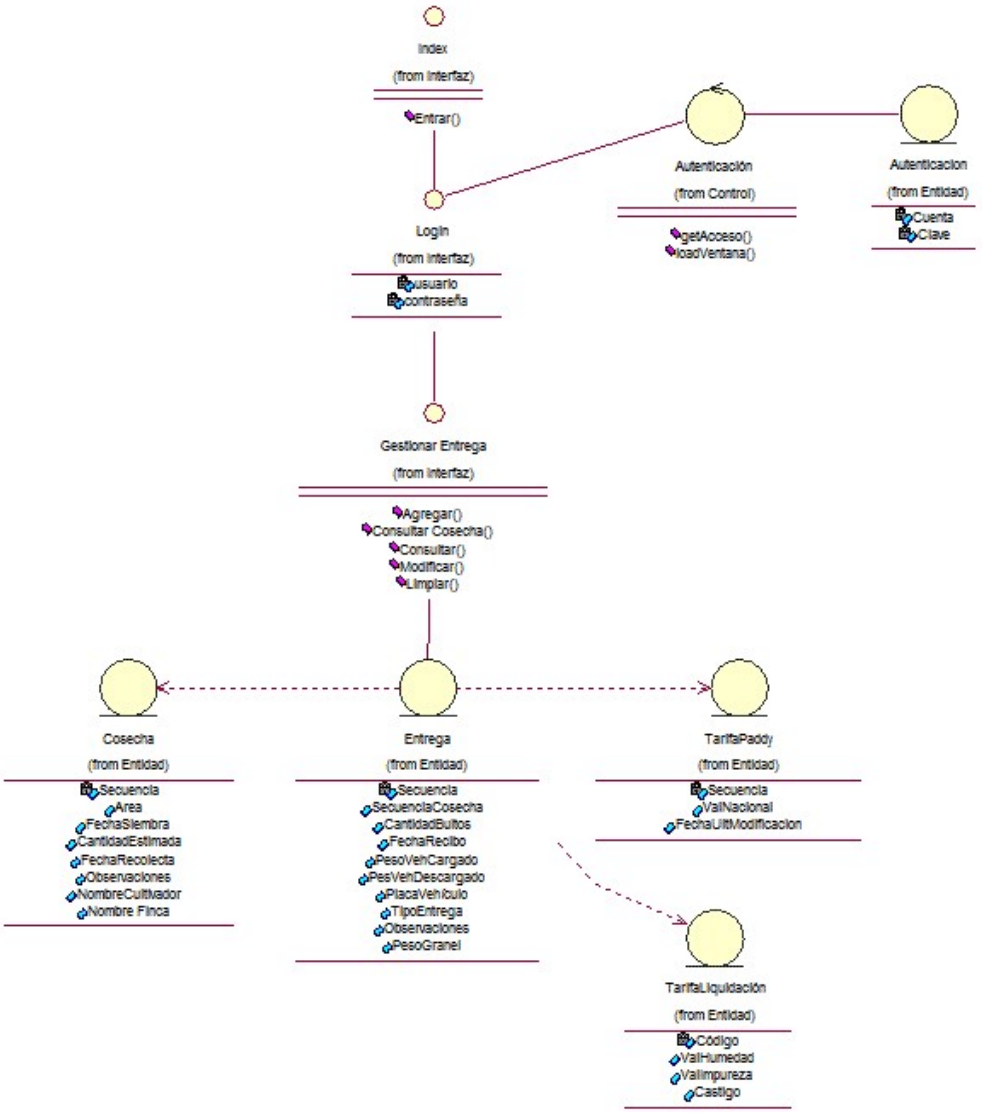
#### 7.2.5.1. Actor Empleado



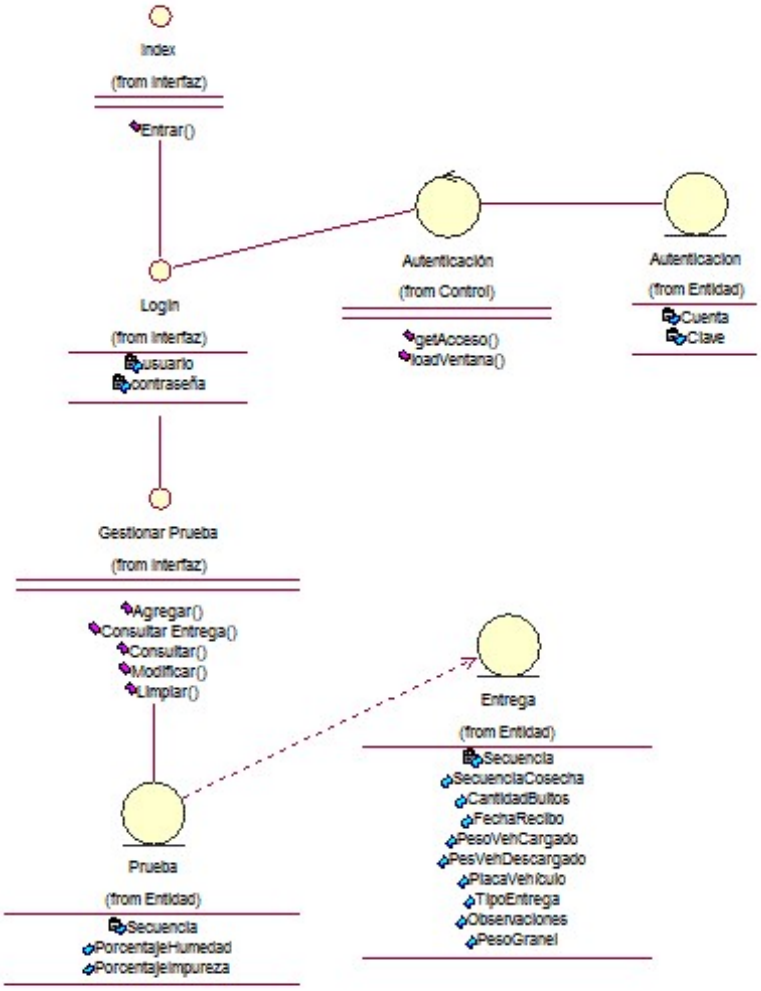
7.2.5.2. Actor Comprador



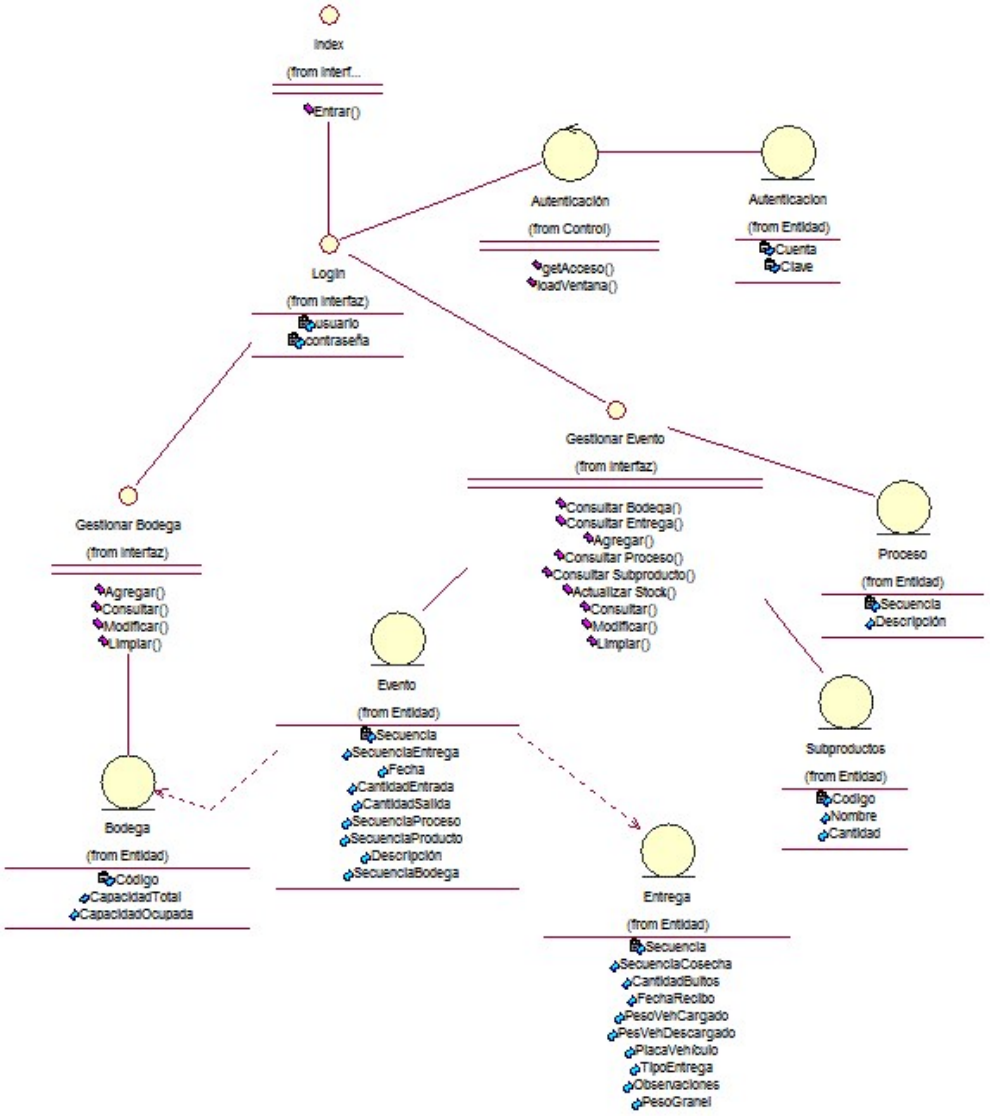
7.2.5.3. Actor Recibidor



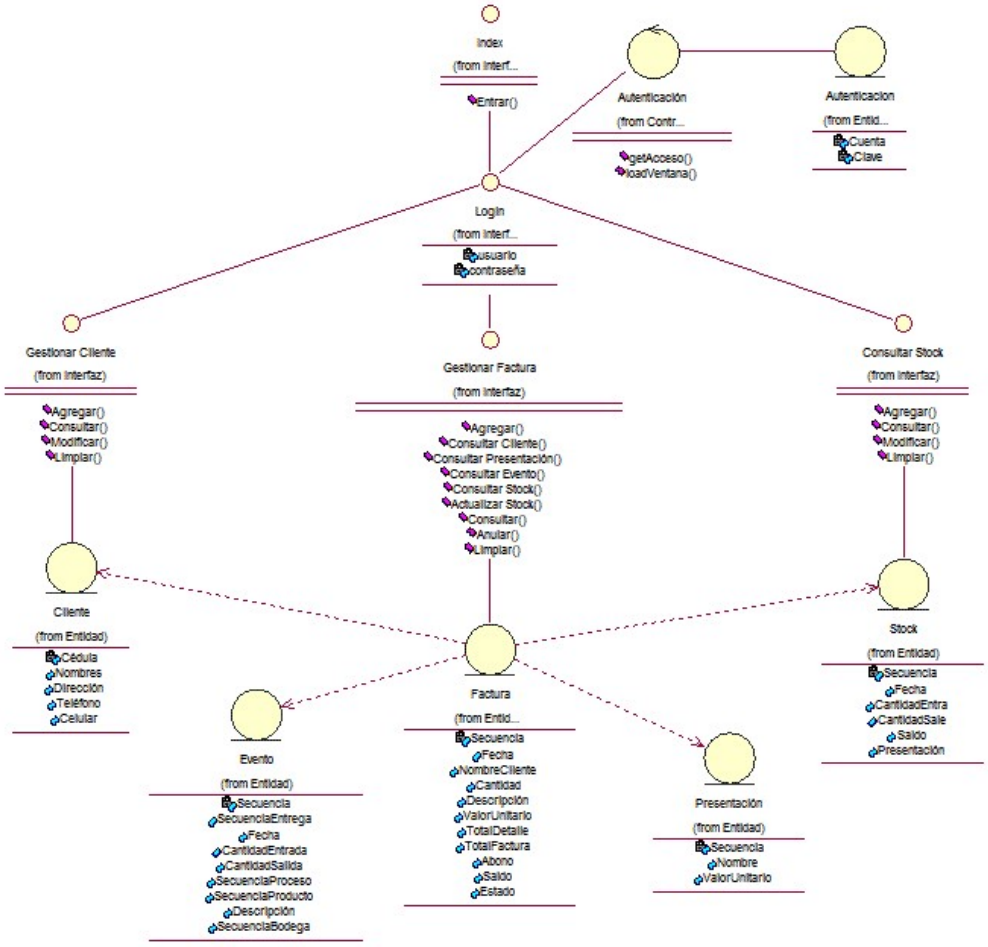
7.2.5.4. Actor Laboratorista



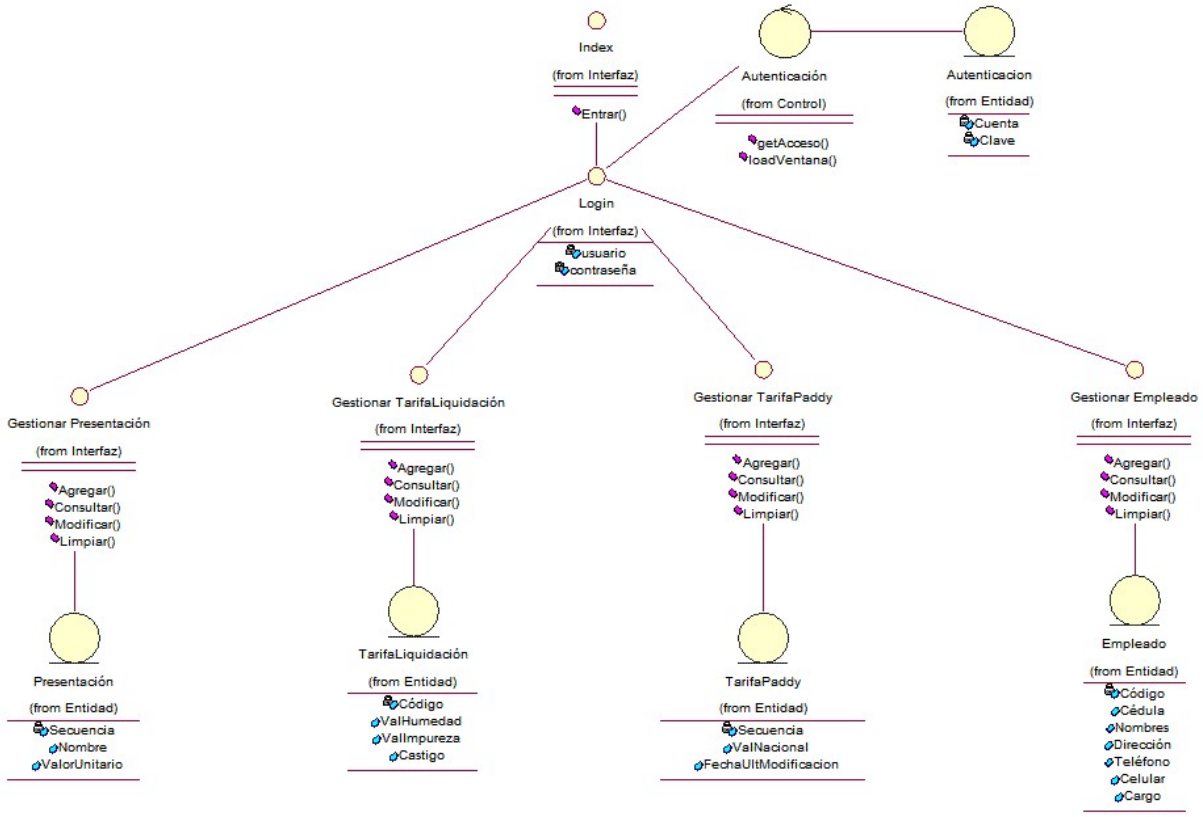
7.2.5.5. Actor Operario



7.2.5.6. Actor Vendedor

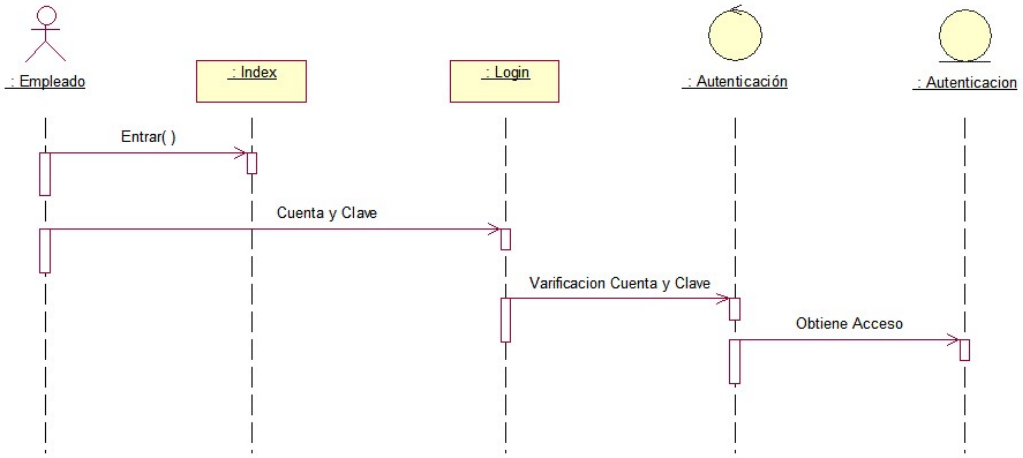


## 7.2.5.7. Actor Administrador



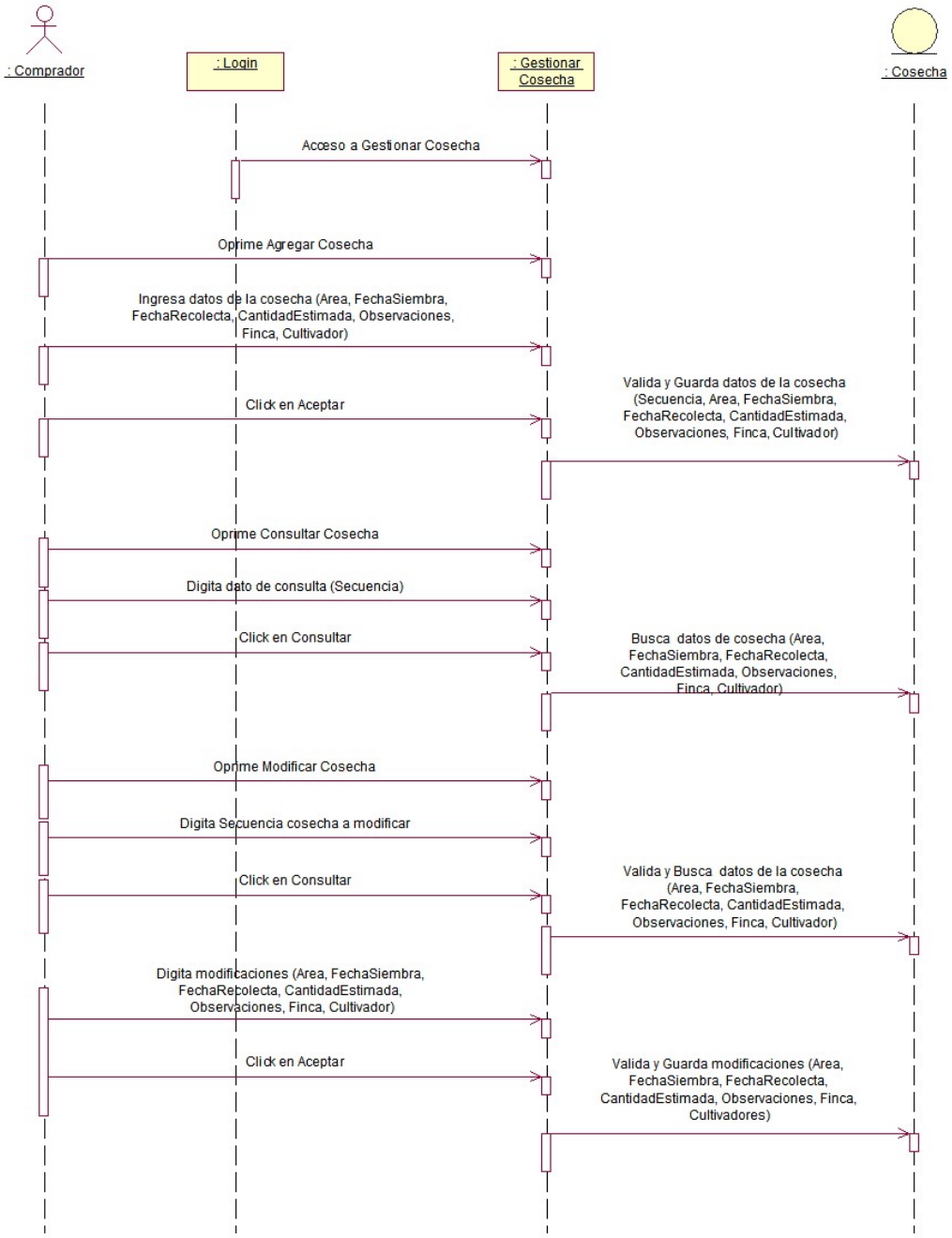
## 7.2.6. Diagramas de Secuencia por Casos de Uso de Inventario

### 7.2.6.1. Actor Empleado

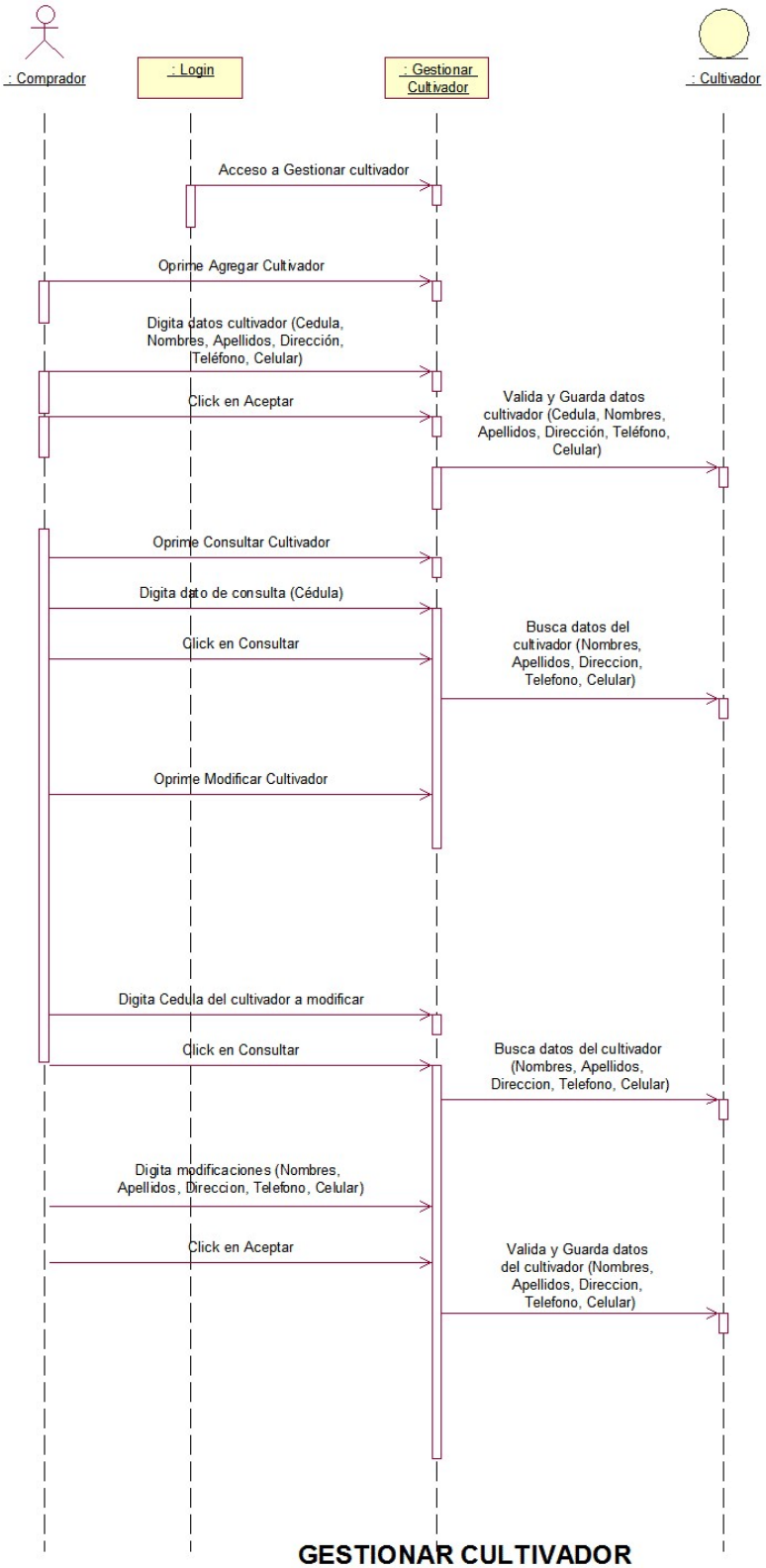



**VALIDAR EMPLEADO**

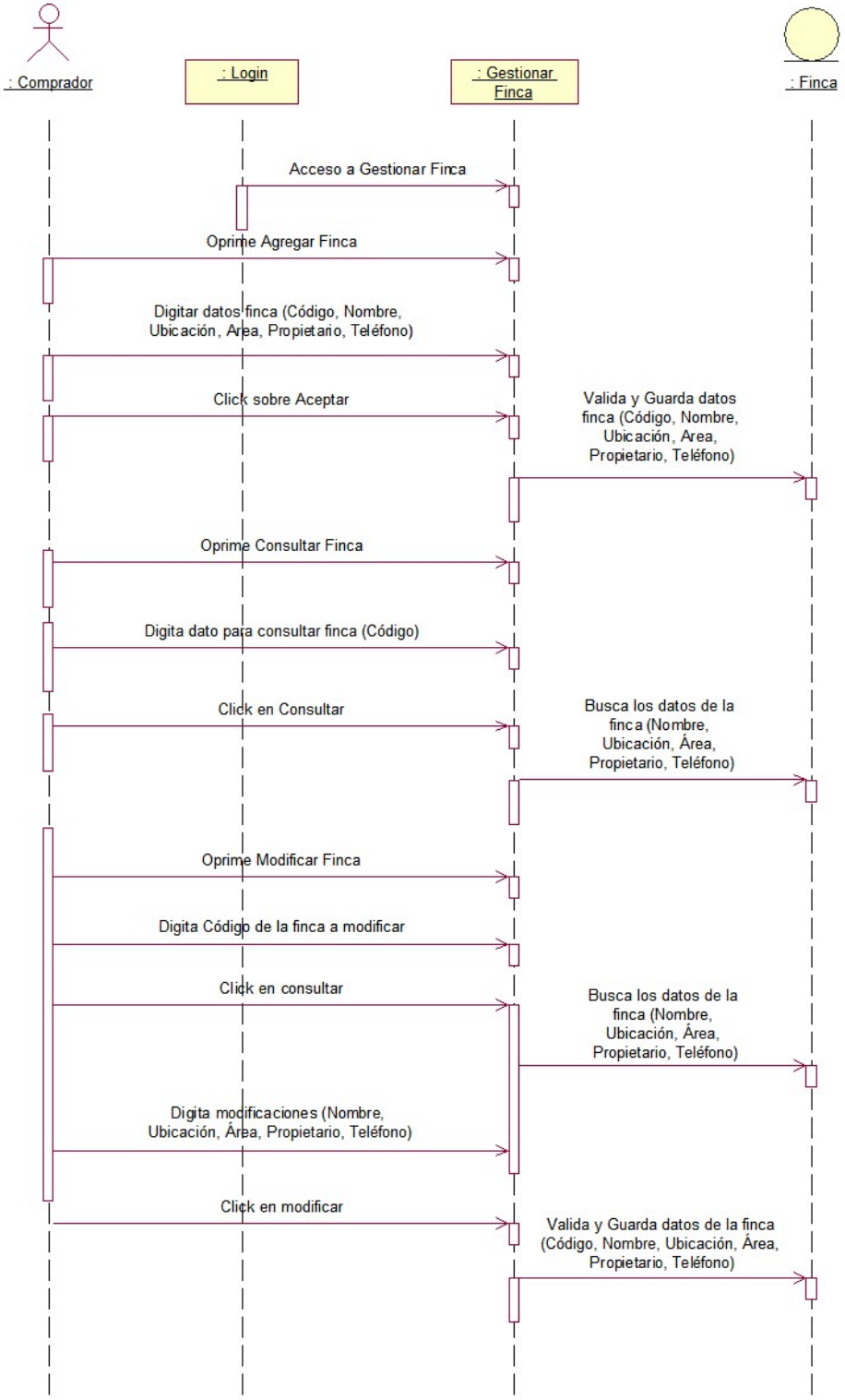
### 7.2.6.2. Actor Comprador




**GESTIONAR COSECHA**

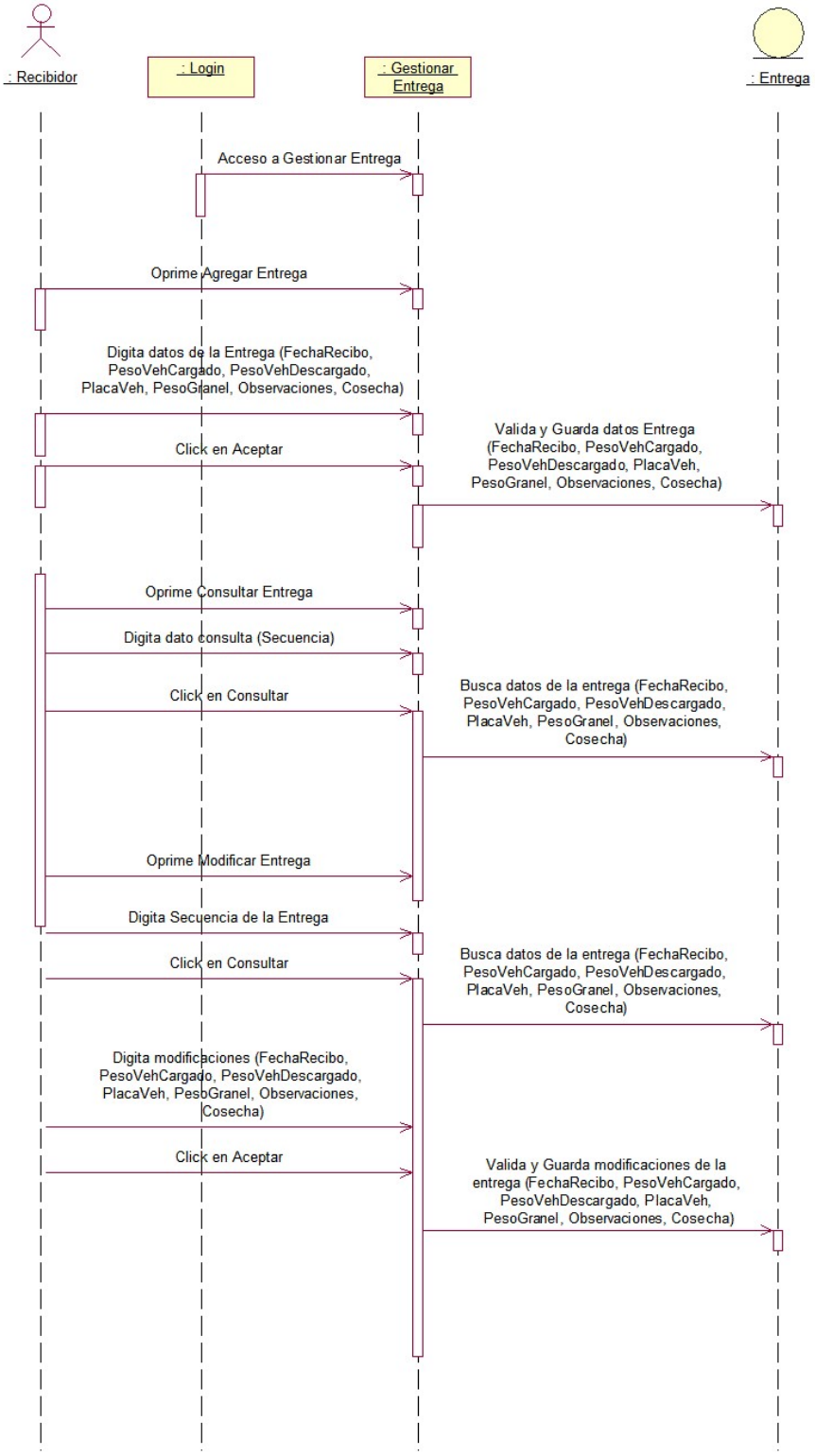



**GESTIONAR CULTIVADOR**

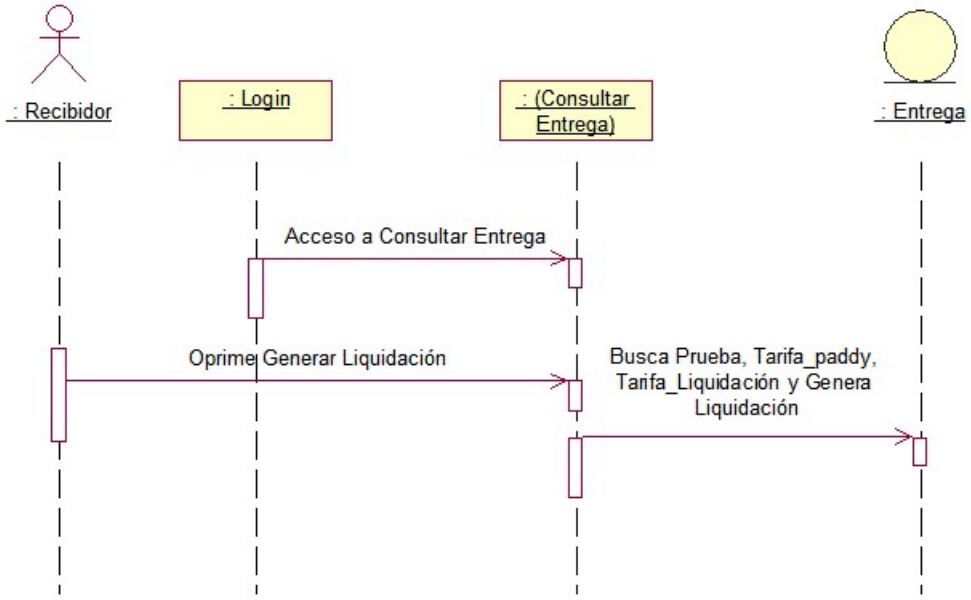



**GESTIONAR FINCA**

### 7.2.6.3. Actor Recibidor

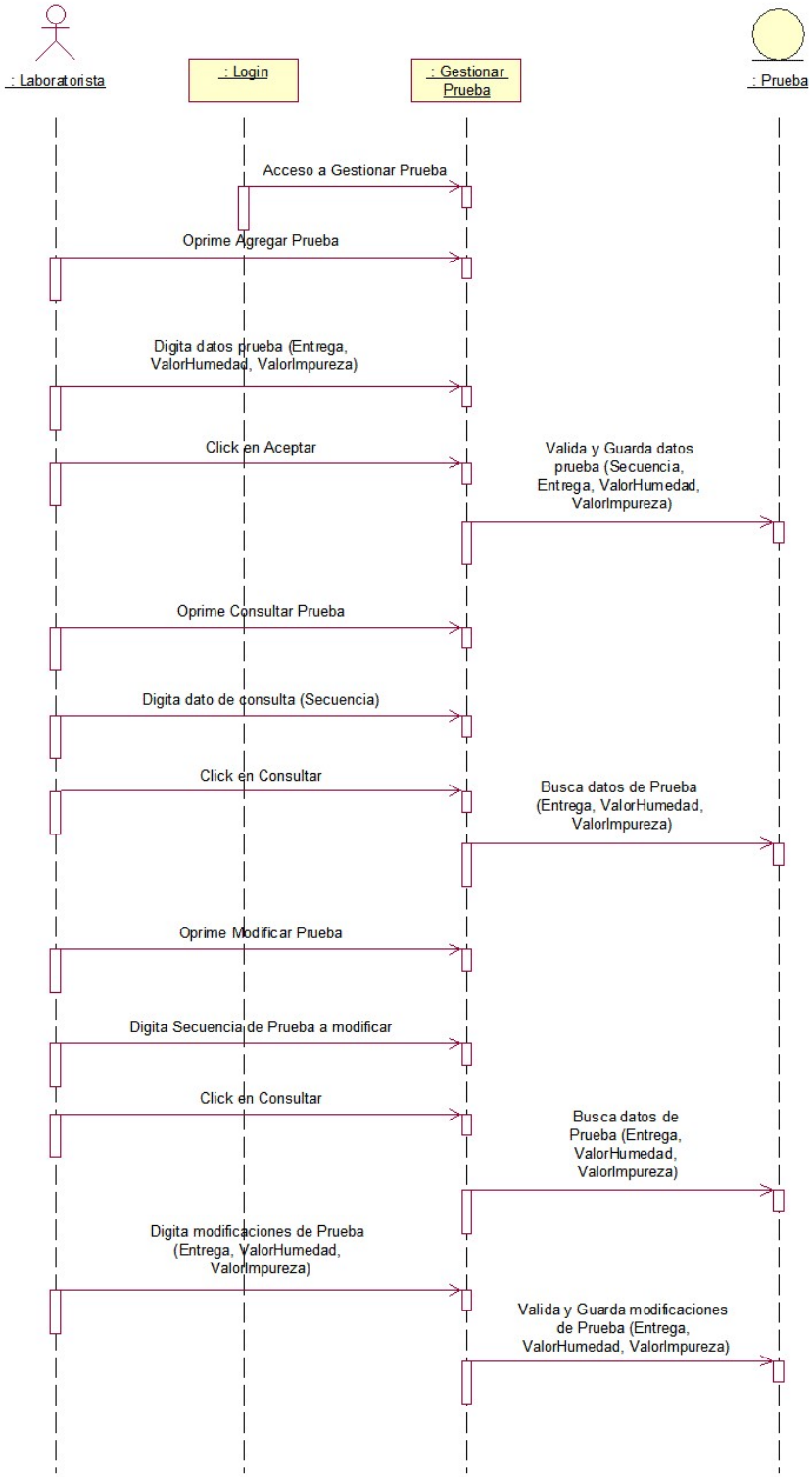



**GESTIONAR ENTREGA**



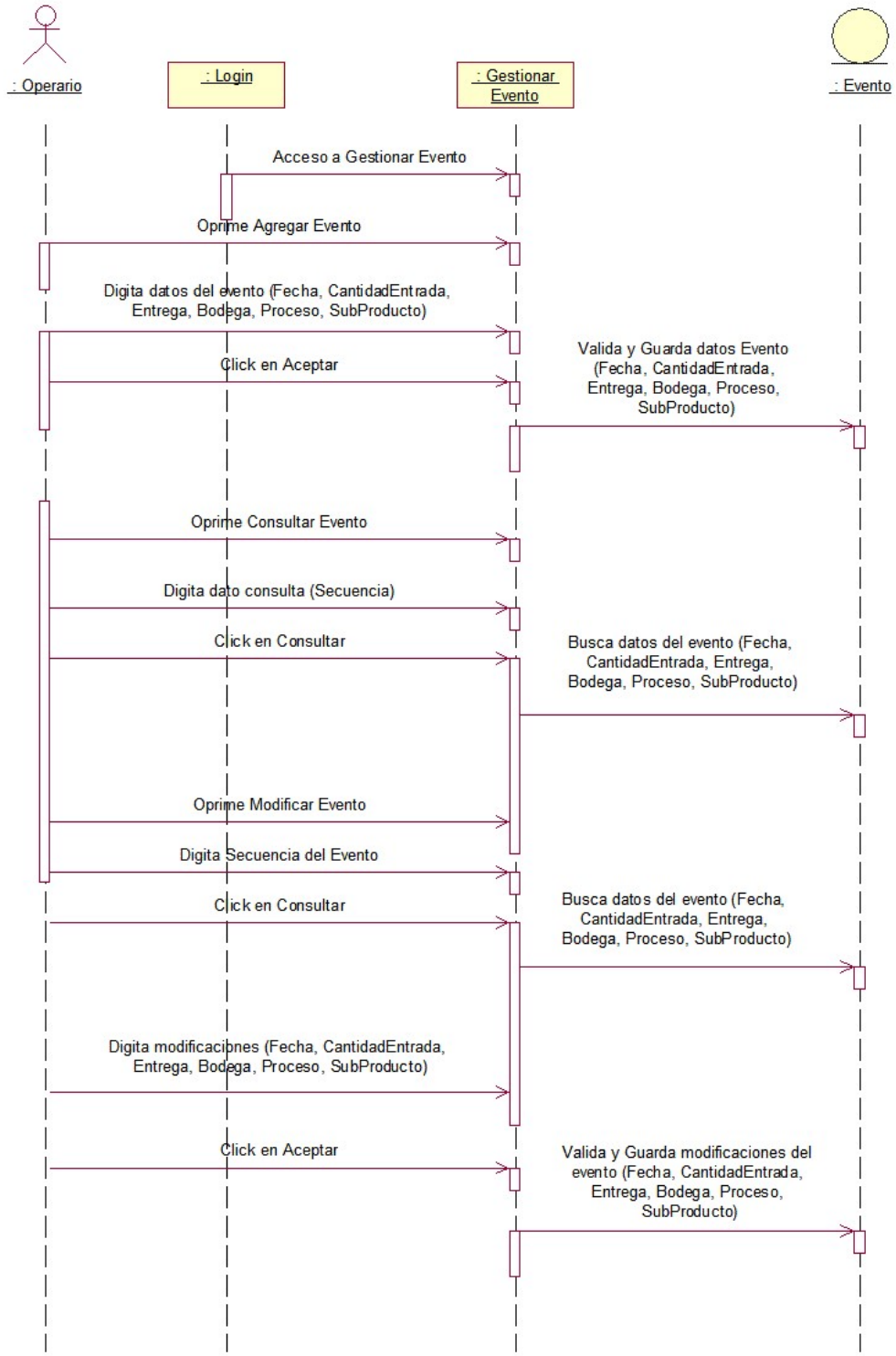

**GENERAR LIQUIDACION**

### 7.2.6.4. Actor Laboratorista

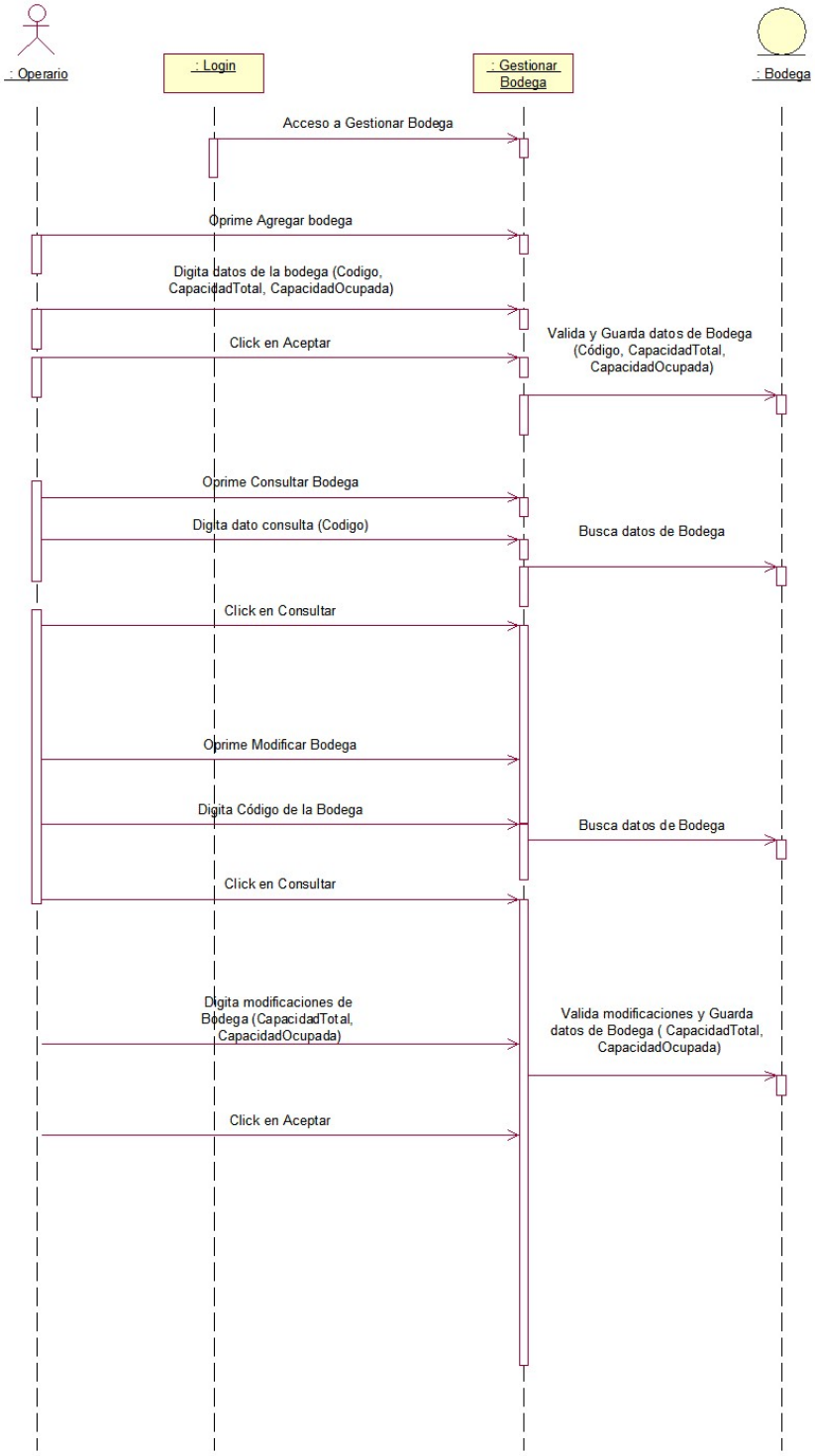



**GESTIONAR PRUEBA**

7.2.6.5. Actor Operario

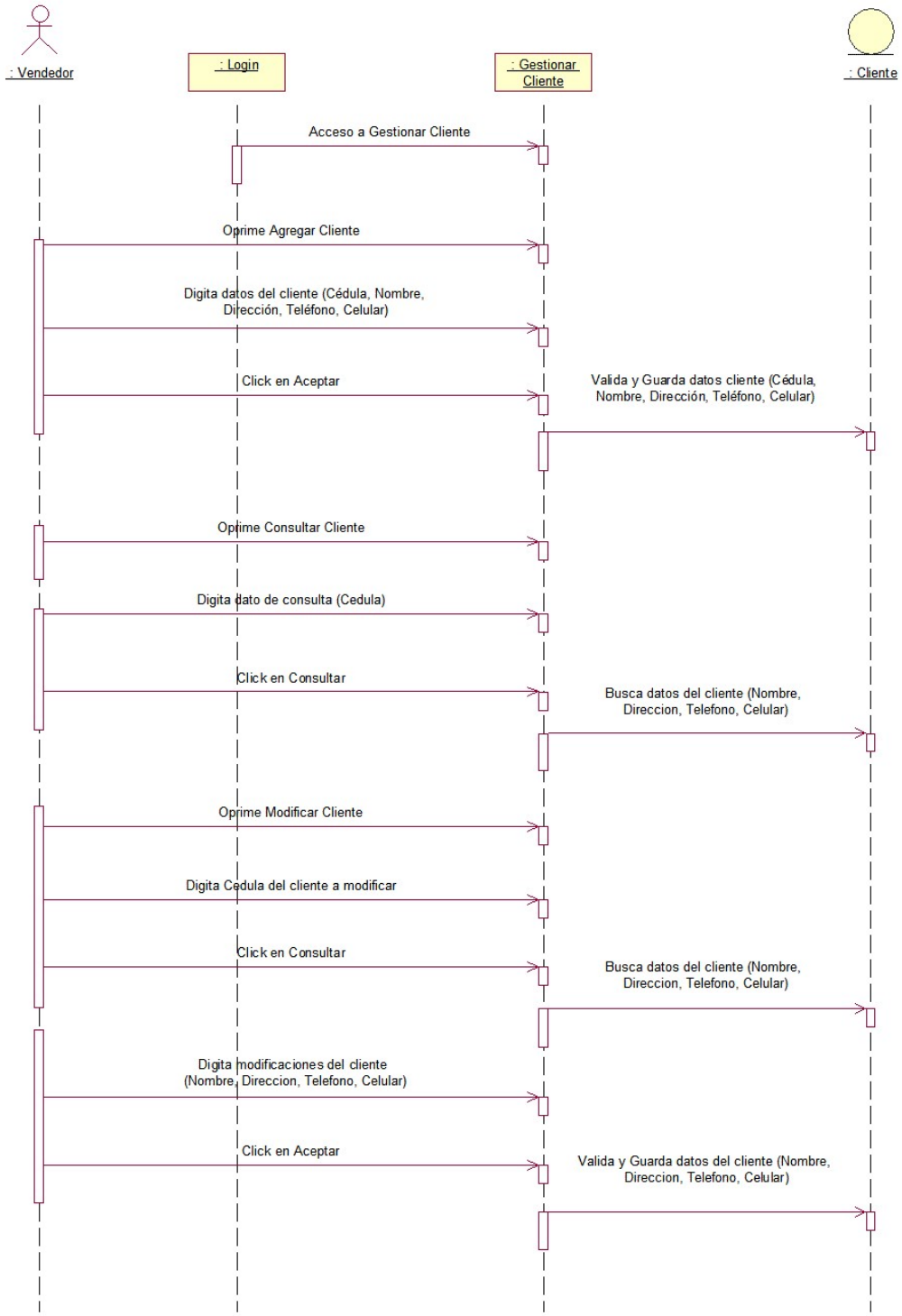



**GESTIONAR EVENTO**

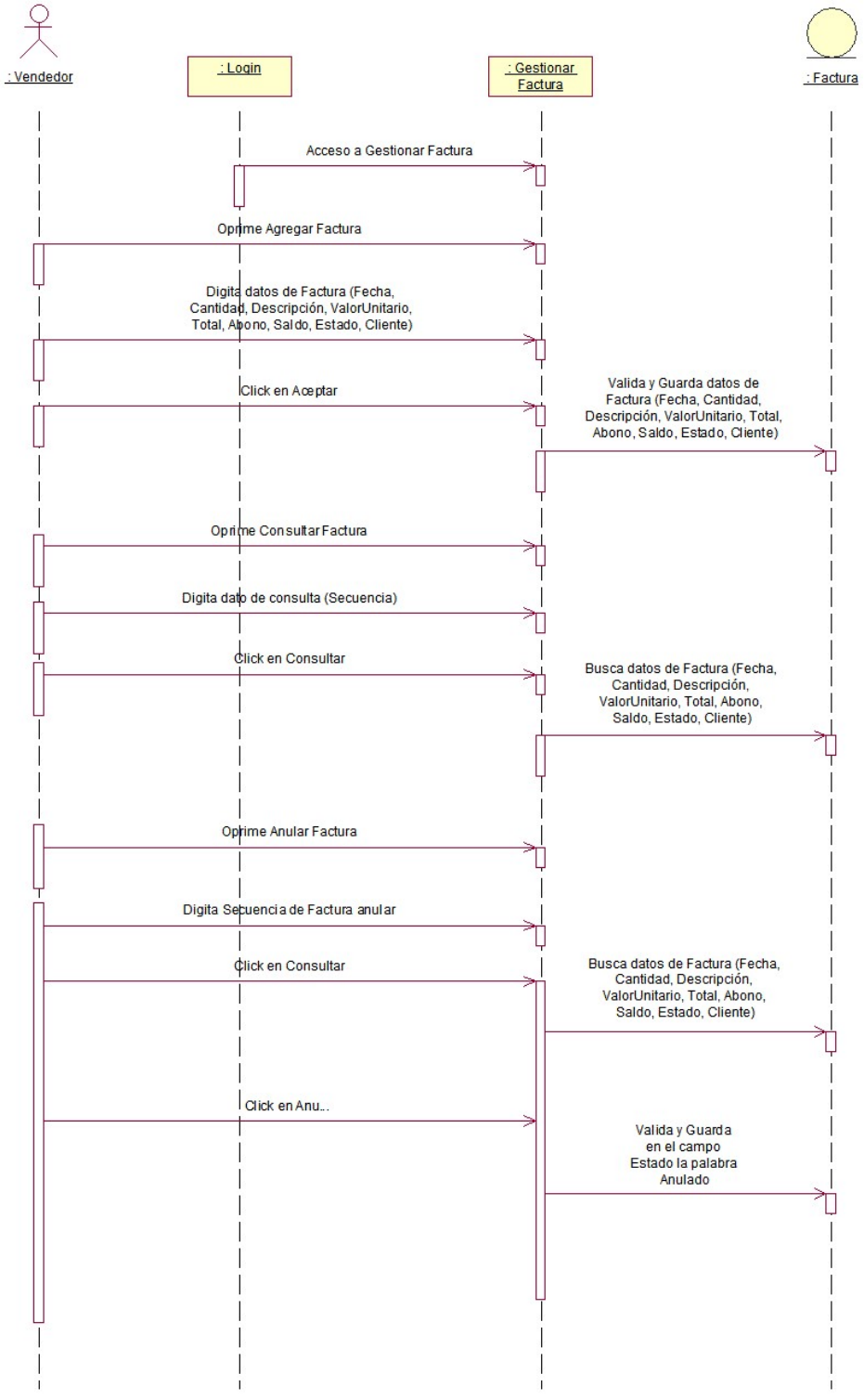



**GESTIONAR BODEGA**

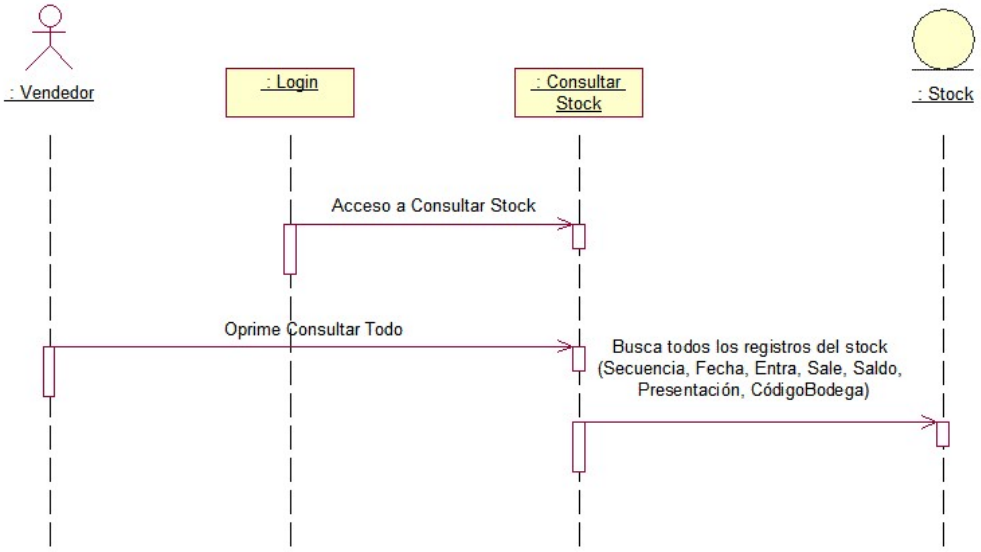
## 7.2.6.6. Actor Vendedor




**GESTIONAR CLIENTE**

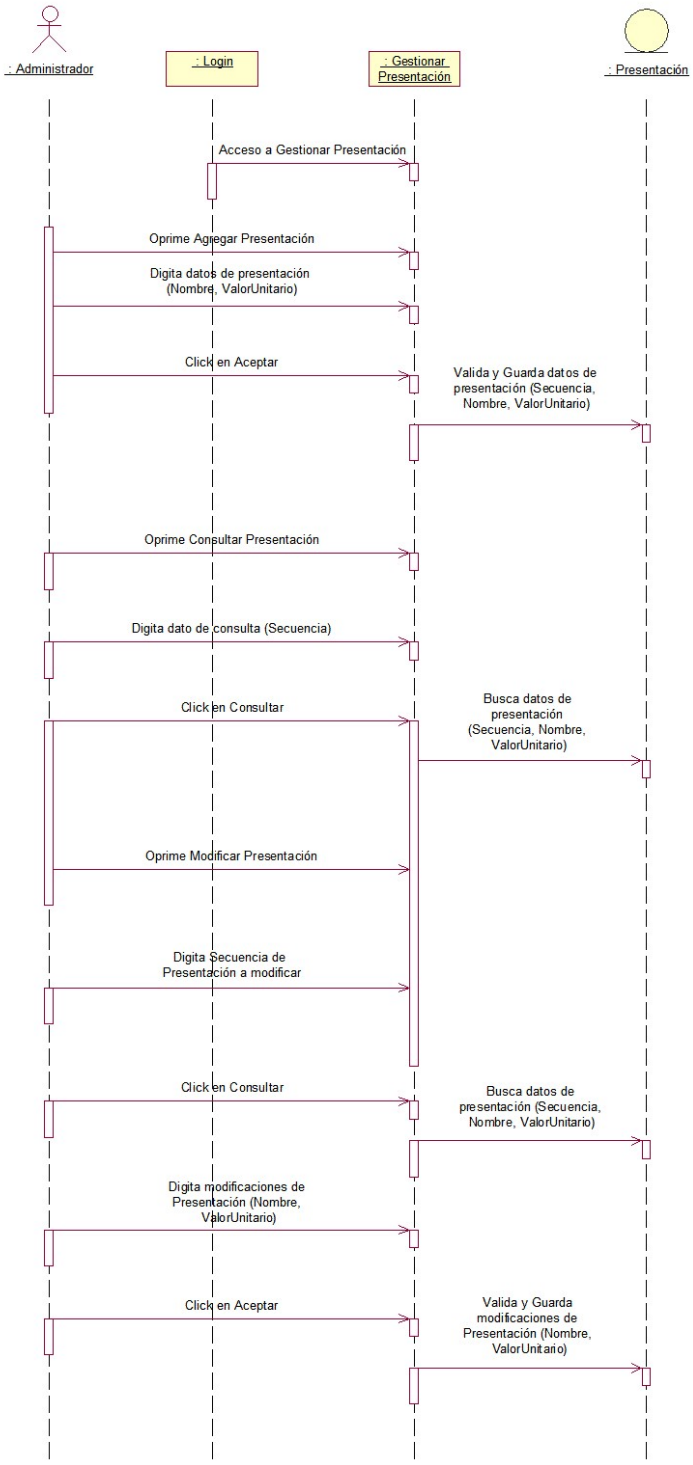



**GESTIONAR FACTURA**

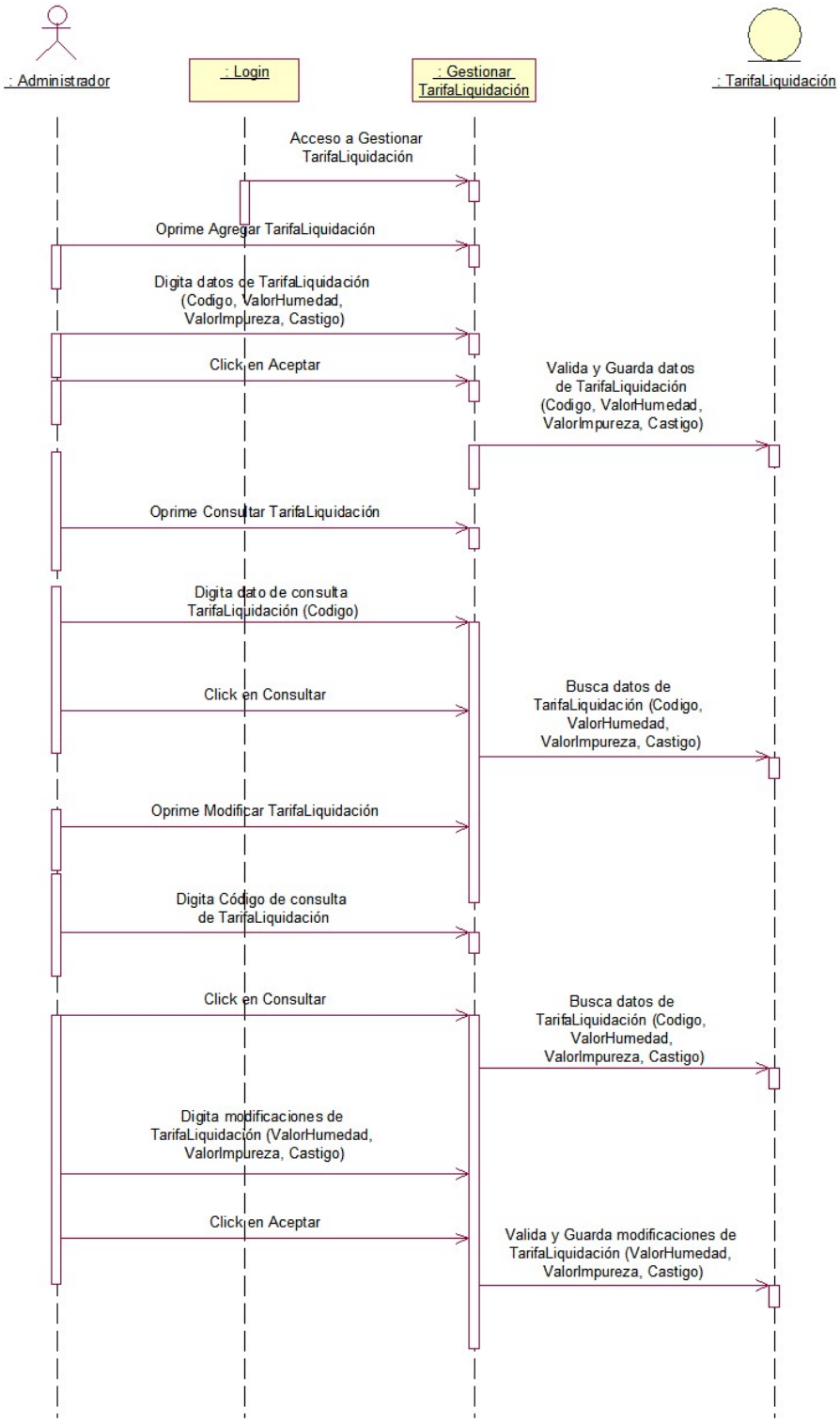



**CONSULTAR STOCK**

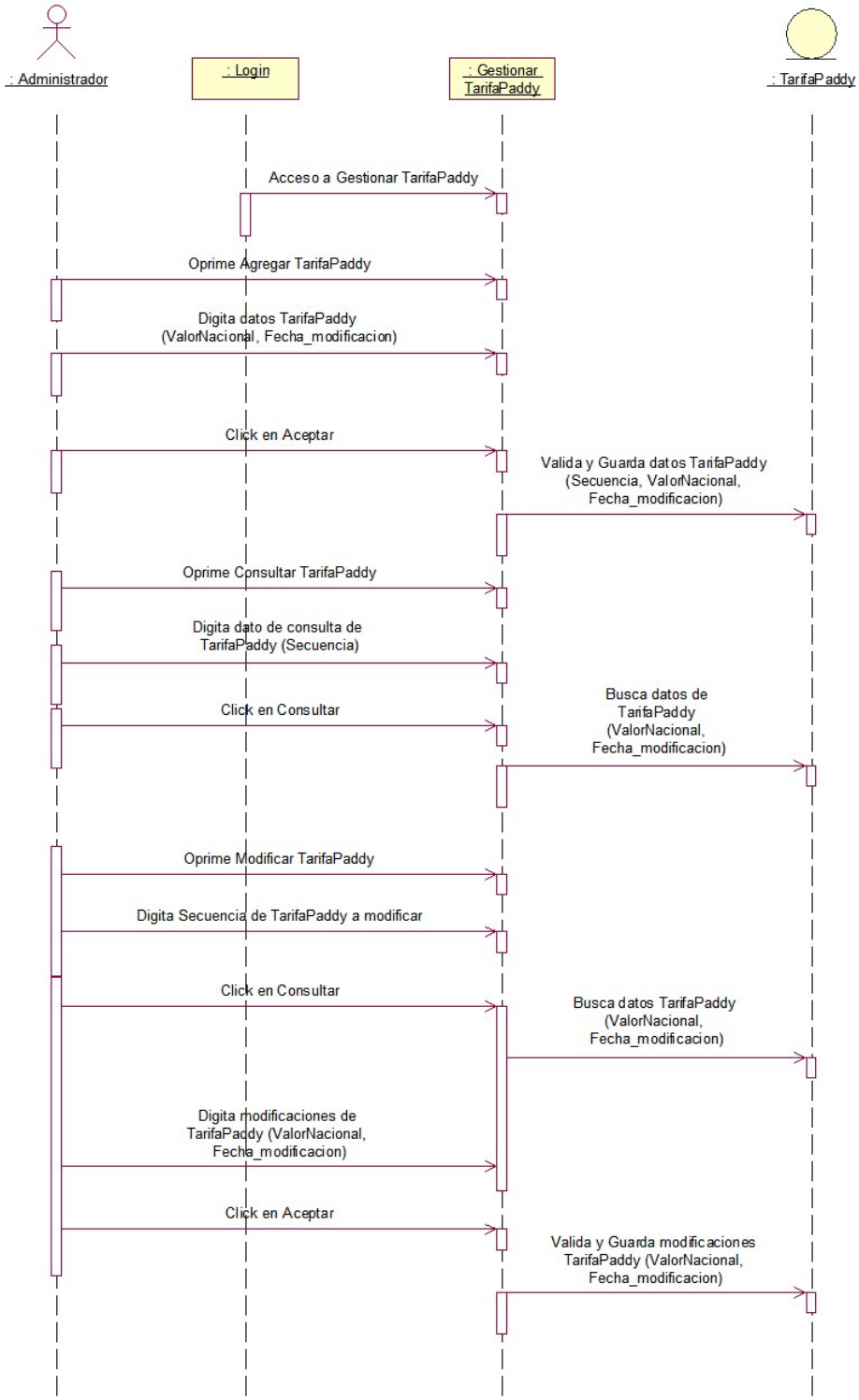
## 7.2.6.7. Actor Administrador



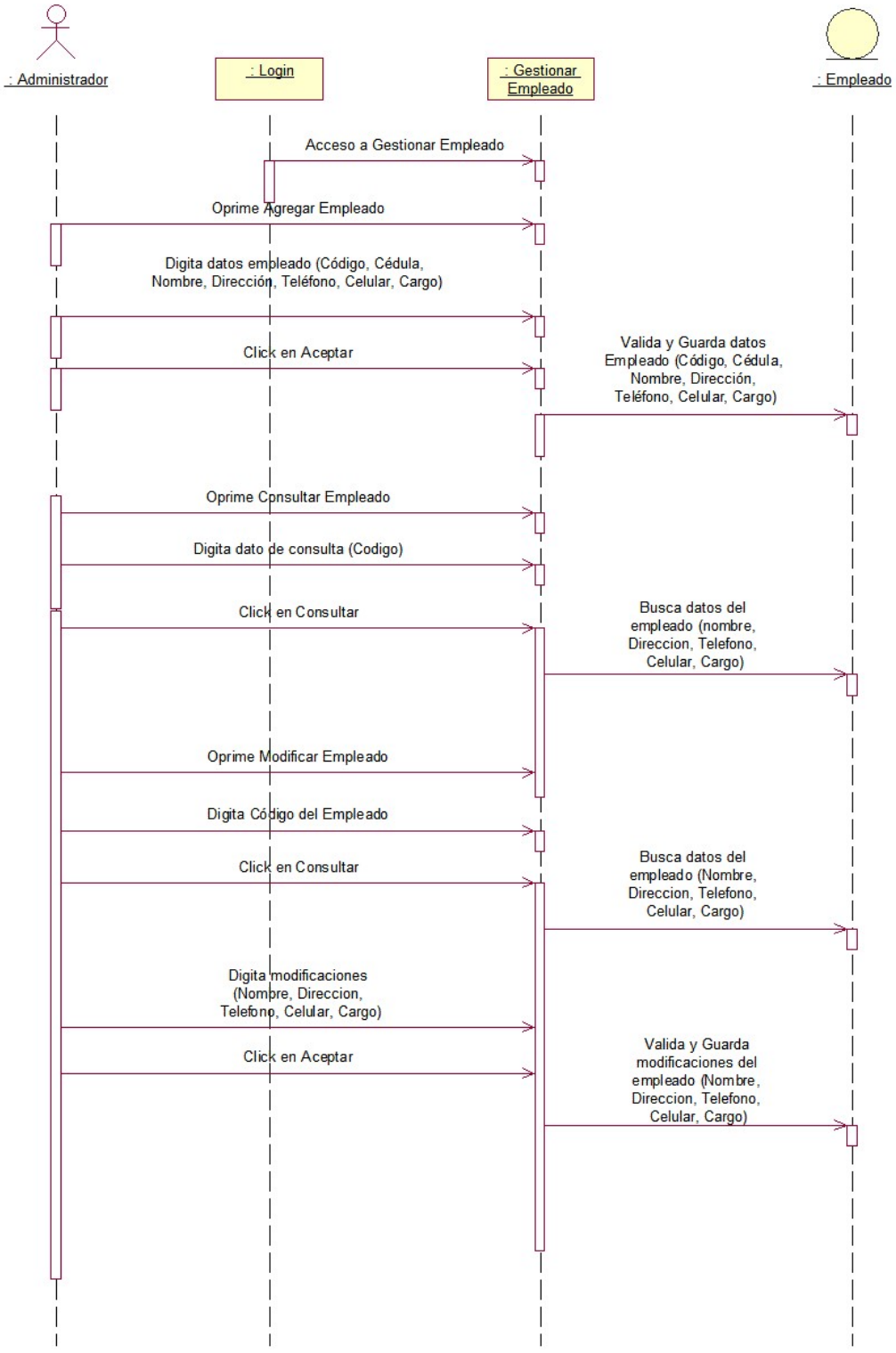

**GESTIONAR PRESENTACION**




**GESTIONAR TARIFA LIQUIDACION**

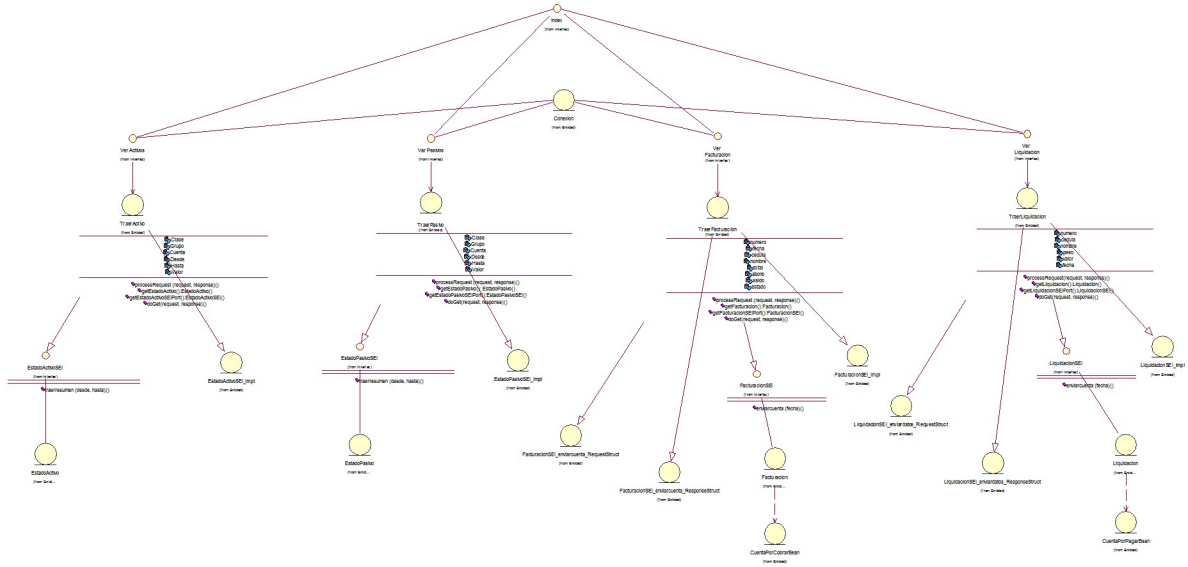



**GESTIONAR TARIFA PADDY**



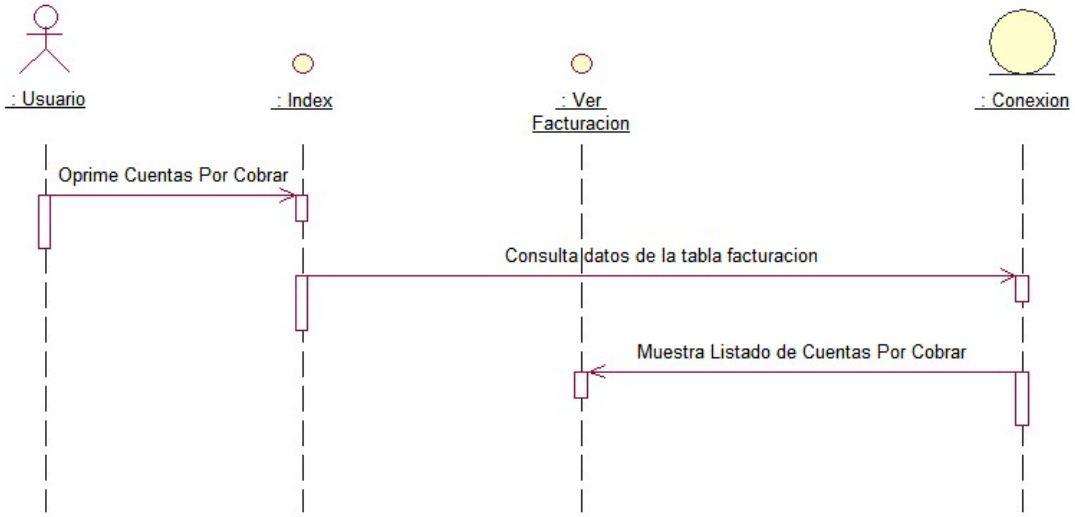

**GESTIONAR EMPLEADO**

### 7.2.7. Diagrama de Clases del Módulo Contabilidad

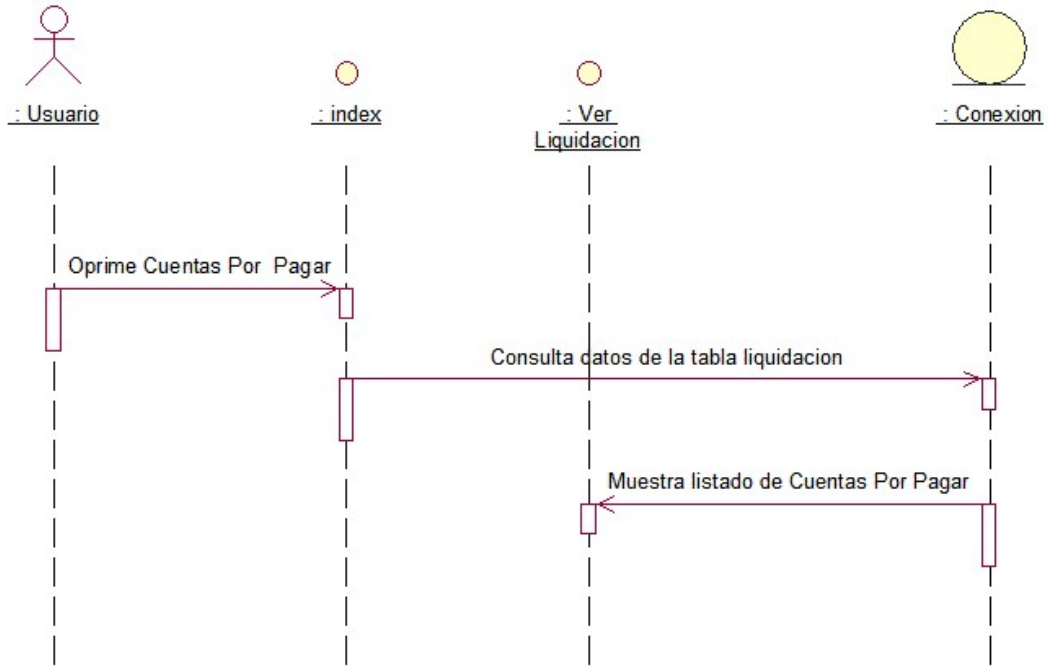


### 7.2.8. Diagrama de Secuencia por Caso de Uso del Módulo Contabilidad

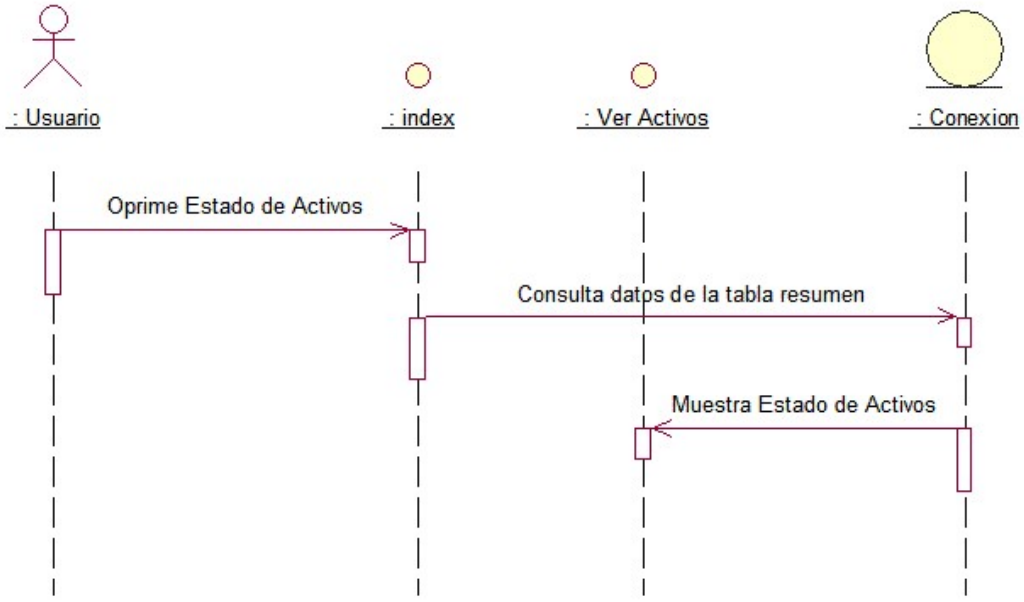
#### 7.2.8.1. Ver Facturación



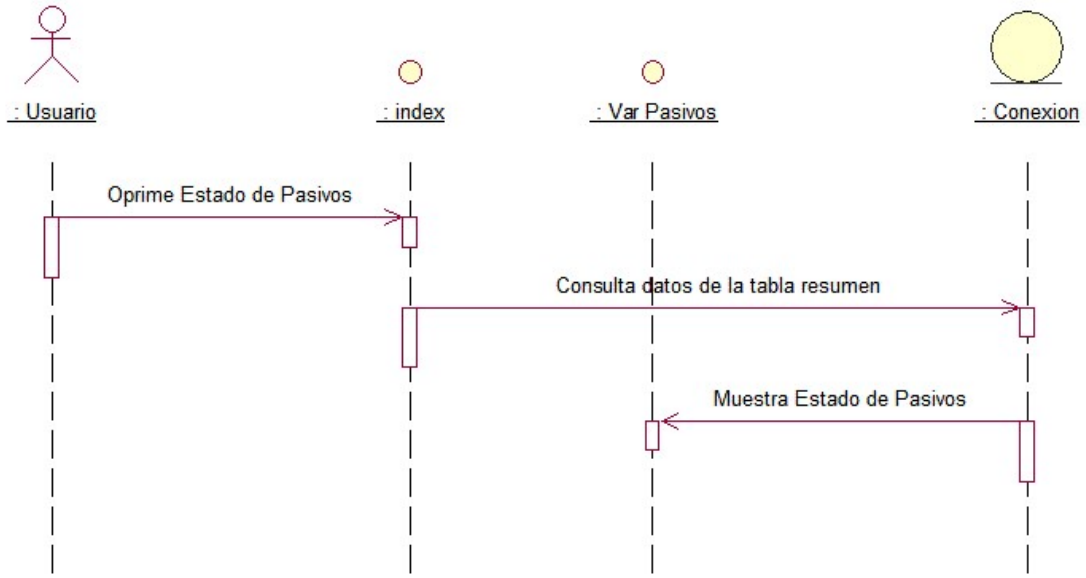
7.2.8.2. Ver Liquidación



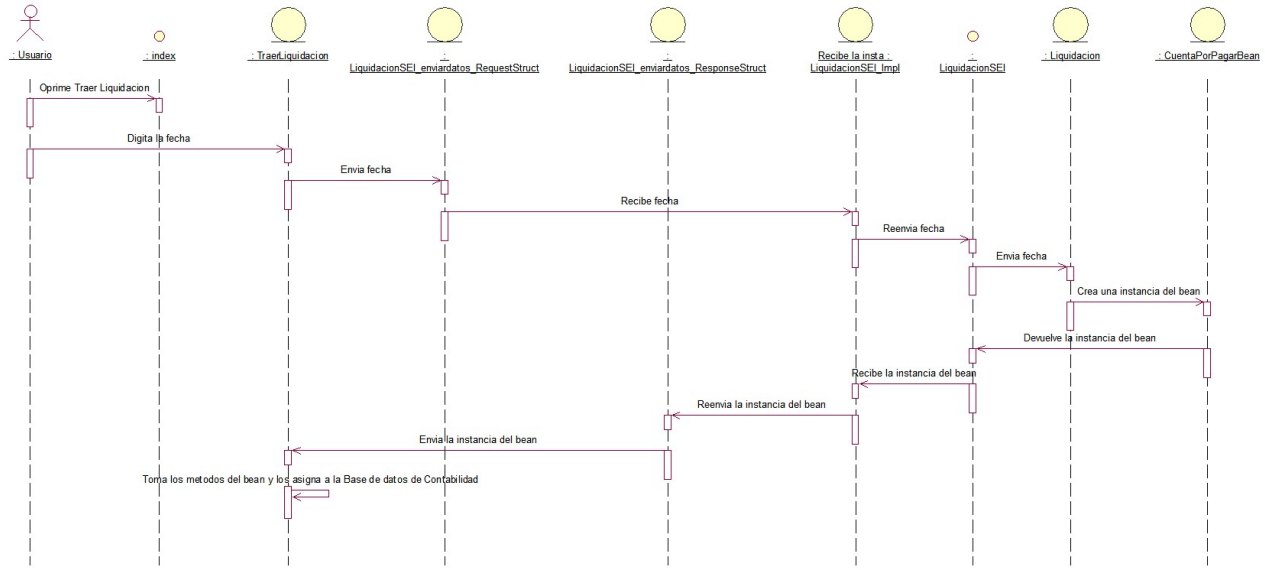

7.2.8.3. Ver Activos



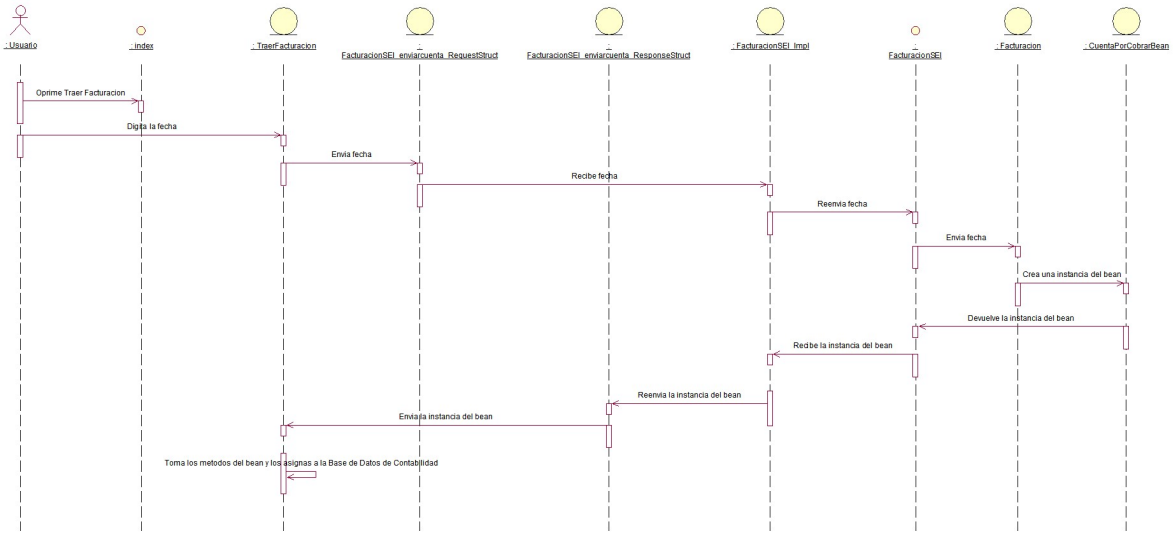
### 7.2.8.4. Ver Pasivos



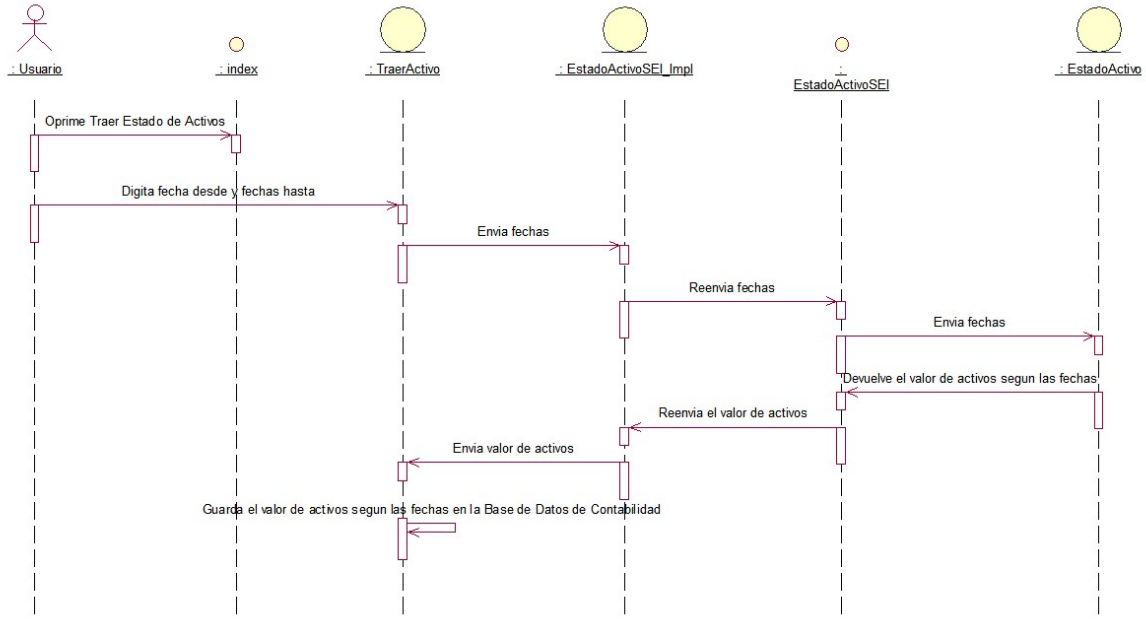

### 7.2.8.5. Traer Liquidación



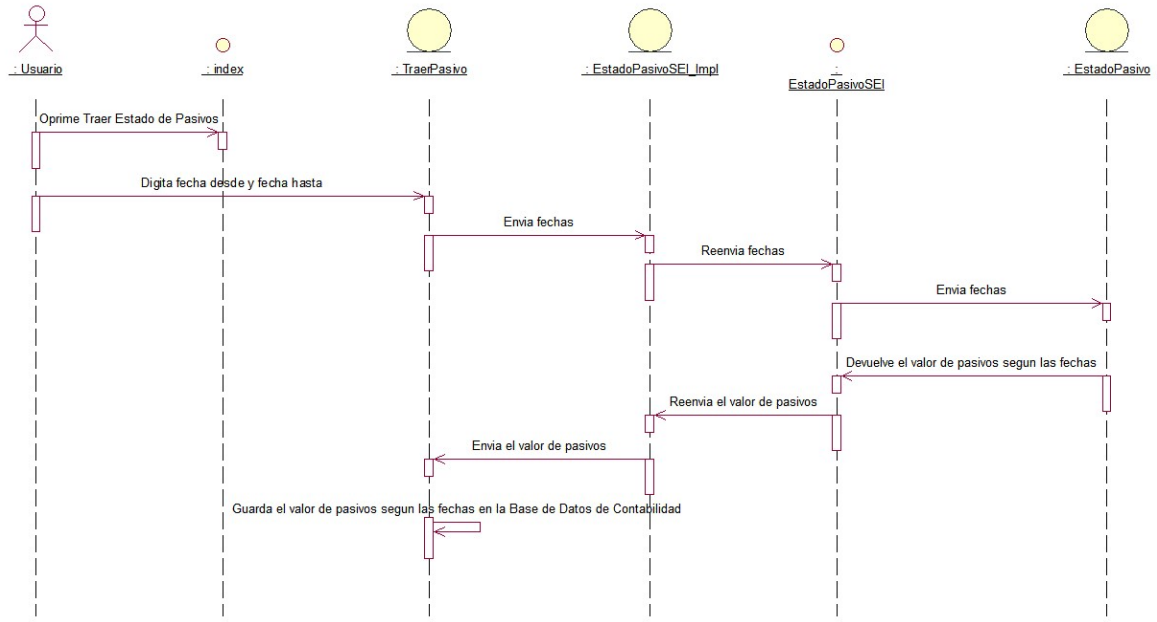
## 7.2.8.6. Traer Facturación




## 7.2.8.7. Traer Estado de Activos



### 7.2.8.8. Traer Estado de Pasivos

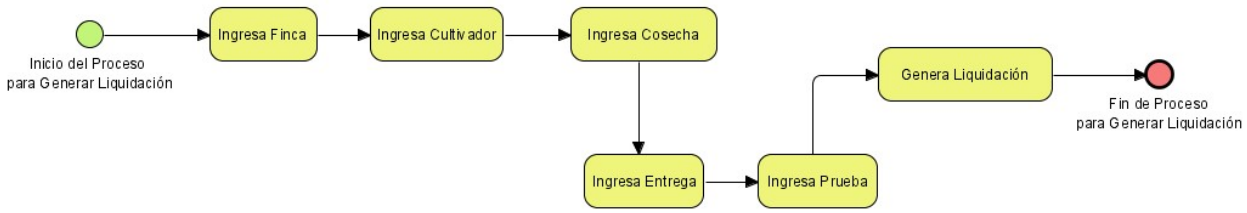



## 7.3. RESULTADOS DE LA FASE DE CONSTRUCCIÓN

### 7.3.1. Diagramas Workflow del Prototipo

#### 7.3.1.1. Diagramas de Workflow del Módulo de Inventario

PROCESO GENERAR LIQUIDACIÓN

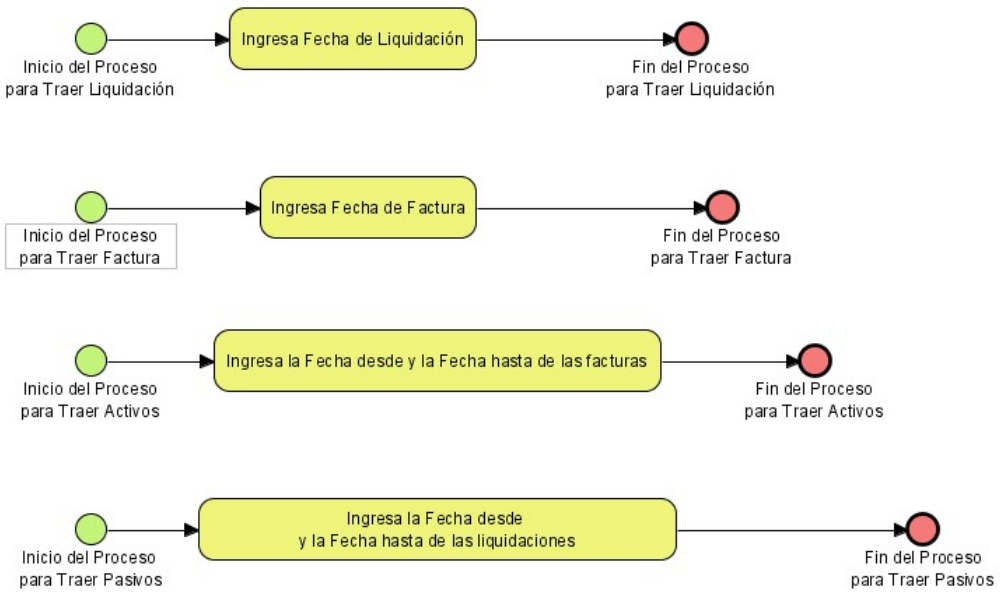


Este proceso inicia cuando el usuario con perfil de Comprador se logea e ingresa a la pantalla de Gestión de Cultivo, allí el Comprador primero debe Ingresar una Finca para lo cual hace clic en el enlace “Agregar Finca”, llena el formulario con los datos de la finca y finalmente da clic en el botón “Agregar”, luego el Comprador debe ingresar un Cultivador para lo cual hace clic en el enlace “Agregar Cultivador”, llena el formulario con los datos del cultivador y finalmente da clic en el botón “Agregar”. Una vez ingresados el Cultivador y la Finca al Sistema, el Comprador debe ingresar la Cosecha para programar el cultivos, para lo cual hace clic en el enlace “Agregar Cosecha”, llena el formulario de la cosecha seleccionando el cultivador y la finca que ingresó anteriormente, luego da clic en el botón “Agregar” para finalizar con su labor como Gestionador de Cultivos. Ahora el usuario con perfil de Recibidor se logea e ingresa a la pantalla de Gestión de Entrega, allí el Recibidor hace clic sobre el enlace “Agregar Entrega”, llena el formulario de la entrega seleccionando la cosecha que ha ingresado el Comprador anteriormente, da clic en el botón “Agregar” para finalizar con su primera parte como Recibidor. Una vez la entrega se encuentre en el sistema, el usuario con perfil de Laboratorista se logea e ingresa a la pantalla de Gestión de Laboratorio, allí el Laboratorista hace clic sobre el enlace “Agregar Prueba” , llena el formulario y automáticamente, en el combo, le aparece la entrega que fue ingresada anteriormente y a la cual se le debe realizar la prueba de laboratorio, selecciona la entrega y da clic en el botón “Agregar”. Nuevamente, el usuario con perfil de Recibidor ingresa a la pantalla de Gestión de Entrega, hace clic en el botón “Generar Liquidación y automáticamente, en el combo, le aparece la entrega ingresada anteriormente puesto que ya tiene una prueba de laboratorio, la selecciona y hace clic en el botón “Generar”; una vez realizada ésta acción al Recibidor se le abrirá otro navegador web con la Liquidación para ser impresa para con esto dar fin al Proceso de Generar Liquidación.

PROCESO AGREGAR FACTURA


Este proceso inicia cuando el usuario con perfil de Vendedor se logea e ingresa a la pantalla de Gestión de Venta, allí el Vendedor primero debe Ingresar un Cliente para lo cual hace clic en el enlace “Agregar Cliente”, llena el formulario con los datos del cliente y finalmente da clic en el botón “Agregar”, luego el Vendedor debe ingresar la factura para lo cual hace clic en el enlace “Agregar Factura”, llena el formulario de la factura seleccionando el cliente que ingresó anteriormente, luego da clic en el botón “Cotizar”, posteriormente en la parte inferior del formulario le aparece la cotización de los insumos que se quieren comprar y un espacio para que se digite el abono que desea dar el cliente una vez realizado esto el Vendedor hace clic en el botón “Generar”; una vez realizada ésta acción al Vendedor se le abrirá otro navegador web con la Factura para ser impresa para con esto dar fin al Proceso de Agregar Factura.

7.3.1.2. Diagrama de Workflow del Módulo de Cliente



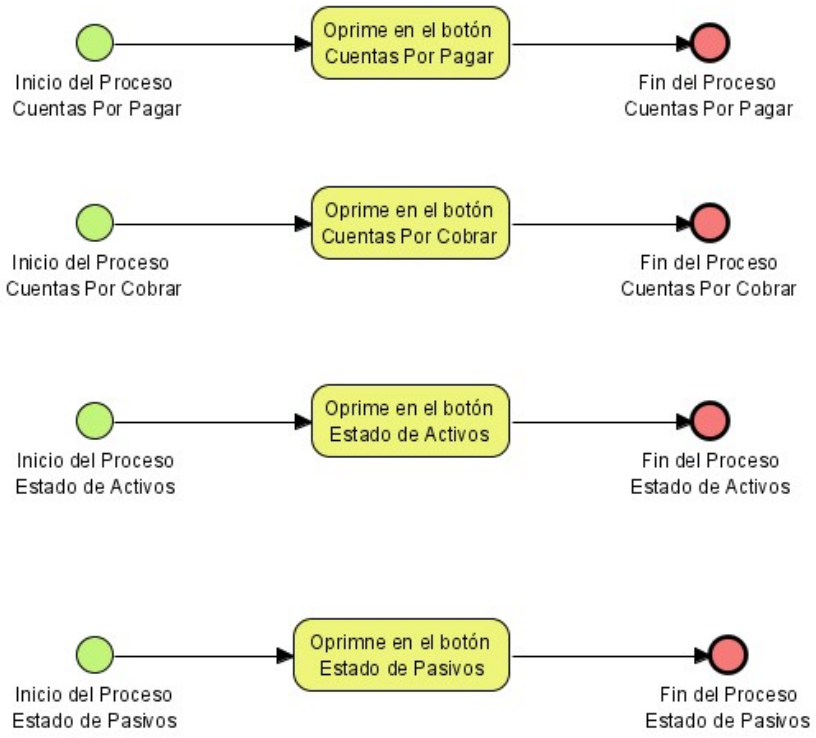

Una vez desplegado el Módulo de Cliente, el usuario podrá realizar las siguientes acciones:

1. Si el usuario hace clic en el enlace “Traer Liquidación” se le cargará una nueva página en la cual digitará la Fecha de la Liquidación que quiera guardar en la Base de Datos de Contabilidad, seguidamente dará clic en el botón “TRAER” y el Web Service realizará la consulta a la Base de Datos de Inventario y la posterior inserción a la Base de Datos de Contabilidad.
2. Si el usuario hace clic en el enlace “Traer Facturación” se le cargará una nueva página en la cual digitará la Fecha de la Factura que quiera guardar en la Base de Datos de Contabilidad, seguidamente dará clic en el botón “TRAER” y el Web Service realizará la consulta a la Base de Datos de Inventario y la posterior inserción a la Base de Datos de Contabilidad.
3. Si el usuario hace clic en el enlace “Traer Estado Activo” se le cargará una nueva página en la cual digitará la Fecha des y la Fecha hasta del rango de Liquidaciones que quiera guardar en la Base de Datos de Contabilidad, seguidamente dará clic en el botón “TRAER” y el Web Service realizará la

consulta a la Base de Datos de Inventario, la suma de los valores de las liquidaciones y la posteriormente realizará la inserción de éste valor a la Base de Datos de Contabilidad como un Activo de la empresa.

4. Si el usuario hace clic en el enlace “Traer Estado Pasivo” se le cargará una nueva página en la cual digitará la Fecha des y la Fecha hasta del rango de Facturas que quiera guardar en la Base de Datos de Contabilidad, seguidamente dará clic en el botón “TRAER” y el Web Service realizará la consulta a la Base de Datos de Inventario, la suma de los saldos de las facturas y la posteriormente realizará la inserción de éste valor a la Base de Datos de Contabilidad como un Pasivo de la empresa.

7.3.1.3. Diagrama de Workflow del Módulo de Contabilidad



Una vez desplegado el Módulo de Contabilidad, el usuario podrá realizar las siguientes acciones:

1. Si el usuario hace clic en el botón “Cuentas Por Pagar” se le cargará una nueva página en la cual se le desplegará un listado de todas las cuentas por pagar ingresadas por medio del Módulo Cliente.
2. Si el usuario hace clic en el botón “Cuentas Por Cobrar” se le cargará una nueva página en la cual se le desplegará un listado de todas las cuentas por cobrar ingresadas por medio del Módulo Cliente
3. Si el usuario hace clic en el botón “Estado de Activos” se le cargará una nueva página en la cual se le desplegará un listado de los activos del rango de fechas ingresados por medio del Módulo Cliente.
4. Si el usuario hace clic en el botón “Estado de Pasivos” se le cargará una nueva página en la cual se le desplegará un listado de los pasivos del rango de fechas ingresados por medio del Módulo Cliente.

7.3.2. Conjunto de Pruebas del Sistema

(Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 4. Implementación y Pruebas > 4.3 Prueba del Sistema).

7.3.3. Conjunto de Pruebas de Integración

(Ver CD > Carpeta Informe Final > Carpeta Plantillas > Archivo frameset.html > Plantillas de Documentación > 4. Implementación y Pruebas > 4.4 Prueba de Integración).

## 7.4. RESULTADOS DE LA FASE DE TRANSICIÓN

### 7.4.1. Manual de Usuario

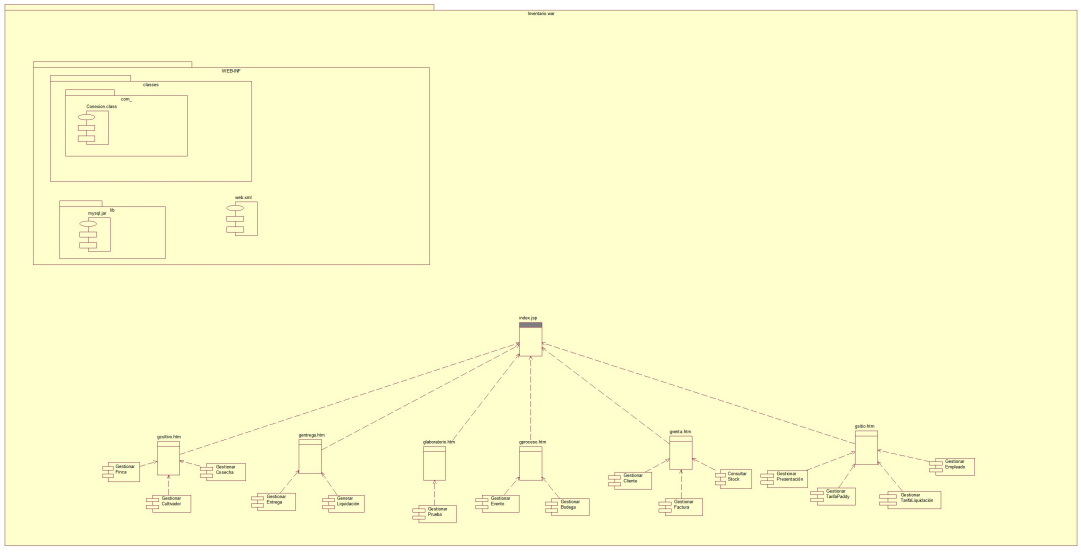
(Ver CD > Carpeta Manuales > Archivo MANUAL_USUARIO.doc).

### 7.4.2. Manual Técnico

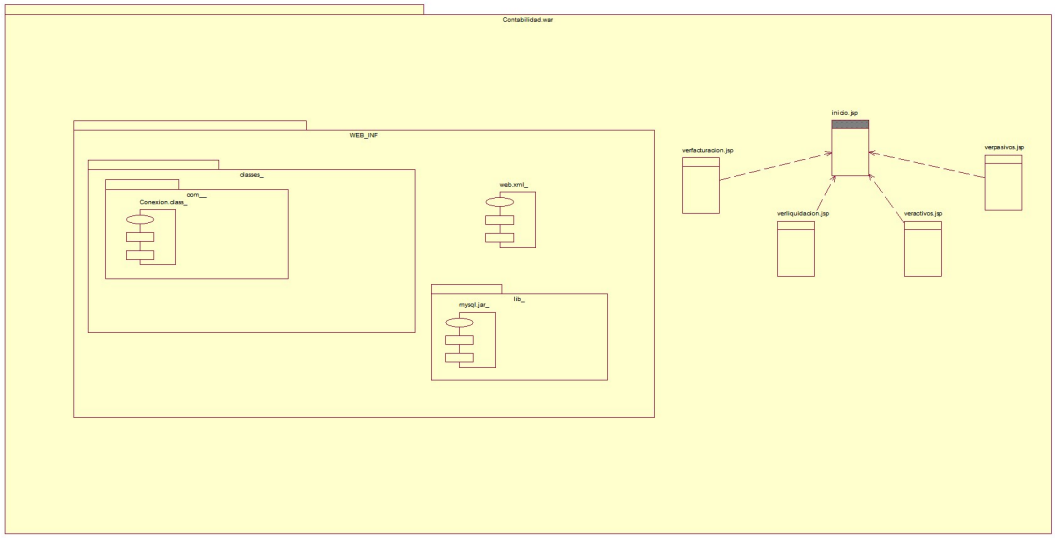
(Ver CD > Carpeta Manuales > Archivo MANUAL_TÉCNICO.doc).

### 7.4.3. Diagrama de Componentes

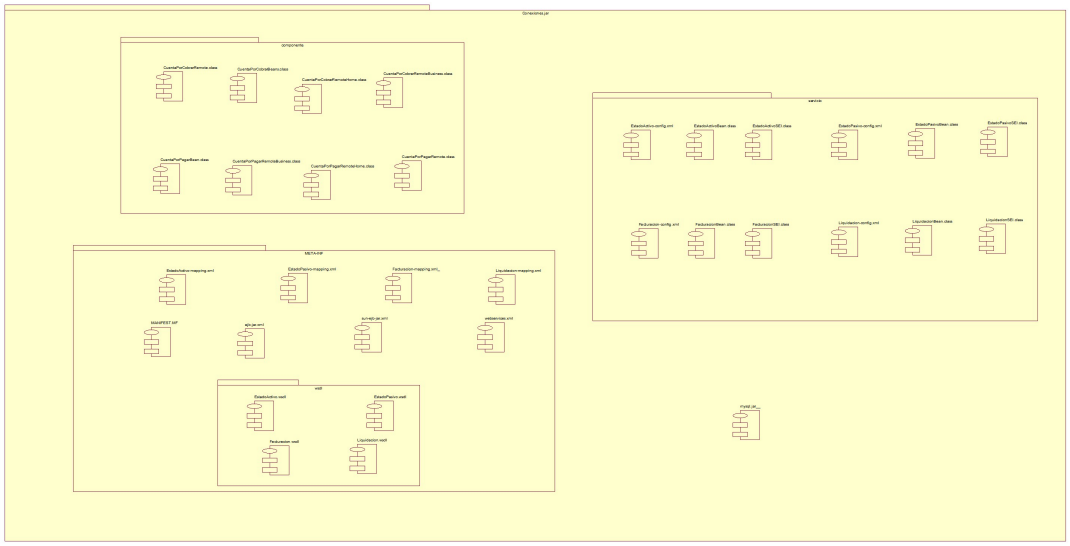
#### 7.4.3.1. Módulo de Inventario



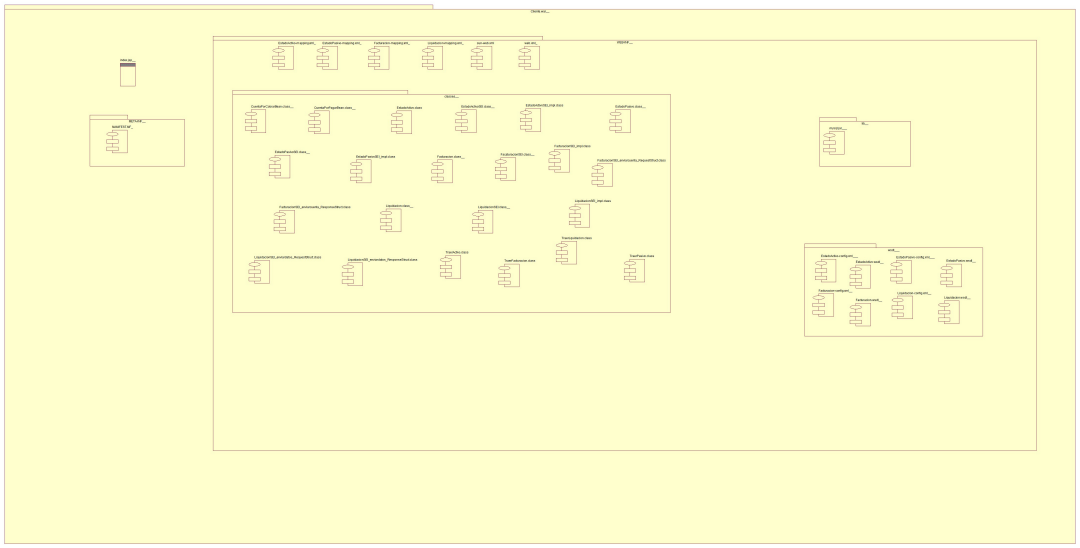
### 7.4.3.2. Módulo de Contabilidad



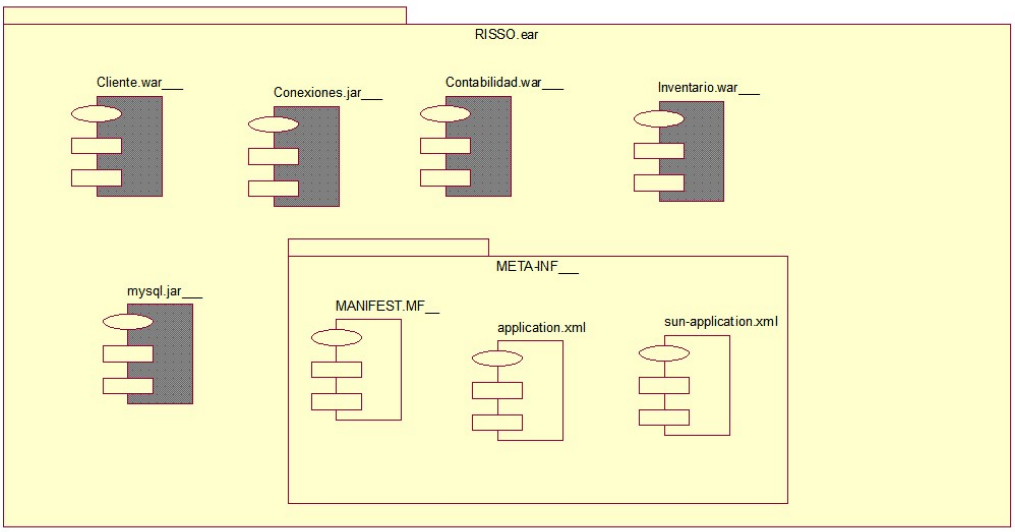

### 7.4.3.3. Módulo de Conexiones



### 7.4.3.4. Módulo de Cliente



### 7.4.3.1. Prototipo RISSO



## 7.4.4. Diagrama de Despliegue del Prototipo RISSO