UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

ANÁLISIS Y DISEÑO DEL SISTEMA DE ATENCIÓN DE TRÁMITES PARA LA COMISIÓN FEDERAL PARA LA PROTECCIÓN CONTRA RIESGOS SANITARIOS

T R A B A J O E S C R I T O

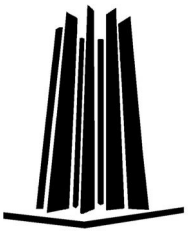
EN LA MODALIDAD DE SEMINARIOS Y CURSOS DE ACTUALIZACIÓN Y CAPACITACIÓN PROFESIONAL QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

RICARDO DAVID SANDOVAL HERNÁNDEZ

ASESOR: M. EN C. MARCELO PÉREZ MEDEL



MÉXICO, 2009

AGRADECIMIENTOS:

A la memoria de Carmen Correa. Te amo.

A mis padres: Rosa y Rogelio por su dedicación y apoyo recibidos, ellos siempre fueron mi guía.

A mi esposa: Fabiola por su amor y su apoyo.

A mi hija: Davia Camila quien es el motor de mi vida.

A todos mis amigos por los buenos y malos momentos compartidos.

A mis profesores por haber contribuido de una u otra forma en mi formación académica.

A Dios por permitirme cumplir esta meta.

TABLA DE CONTENIDOS.

1. CONTEXTO.

1.1. ANTECEDENTES.

La Comisión Federal para la Protección Contra Riesgos Sanitarios (COFEPRIS), es un órgano desconcentrado de la Secretaría de Salud creado el 5 de julio del 2001, que cuenta con autonomía administrativa, técnica y operativa, cuya misión es la de proteger a la población contra riesgos sanitarios, mediante su evaluación y la aplicación oportuna y eficaz de las medidas de intervención necesarias para prevenirlos, minimizarlos o eliminarlos, a partir de una adecuada comunicación de riesgos a la población que facilite la protección de su salud, y con la visión de lograr una sociedad sana debidamente protegida contra riesgos sanitarios.

Un riesgo sanitario es la probabilidad de ocurrencia de un evento exógeno adverso, conocido o potencial que ponga en peligro la salud o la vida humana, derivada de la exposición involuntaria de la población a factores biológicos, químicos o físicos presentes en los productos, servicios o publicidad, en el medio ambiente o en el lugar de trabajo. Vale la pena mencionar que ninguna actividad humana está libre de riesgos (no existe riesgo cero) y los efectos del riesgo dependerán del factor de riesgo, la dosis, el tiempo y la frecuencia de la exposición, así como de la susceptibilidad individual.

La COFEPRIS adopta un sistema de trabajo basado en procesos, por lo que se divide en varias áreas técnicas que en realidad son las diferentes comisiones que la conforman y asimismo sus diferentes departamentos de éstas y un Centro Integral de Servicios (CIS) que es el encargado de tratar con el usuario en cuanto a la recepción y entrega de trámites; el flujo de un trámite es que ingresa a la Comisión a través del CIS, éste se encarga de turnarlo al área correspondiente, el área resuelve y genera una resolución y la entrega al CIS, y éste es el encargado de entregar al usuario la resolución de su trámite.

1.2 ATRIBUCIONES Y FUNCIONES.

Atribuciones, funciones y características que por Ley son determinadas a la COFEPRIS:

- El control y vigilancia de los establecimientos de salud.
- La prevención y el control de los efectos nocivos de los factores ambientales en la salud del hombre.
- La salud ocupacional y el saneamiento básico.
- El control sanitario de productos, servicios y de su importación y exportación, y de los establecimientos dedicados al procesos de los productos.
- El control sanitario del proceso, uso, mantenimiento, importación, exportación y disposición final de equipos médicos, prótesis, órtesis, ayudas funcionales, agentes de diagnóstico, insumos de uso odontológico, materiales quirúrgicos, de curación y productos higiénicos, y de los establecimientos dedicados al proceso de los productos.
- El control sanitario de la publicidad de las actividades, productos y servicios.

- El control sanitario de la disposición de órganos, tejidos y sus componentes y células de seres humanos.
- La sanidad internacional.
- El control sanitario de las donaciones y trasplantes de órganos, tejidos células de seres humanos.

La COFEPRIS se encuentra al frente de un comisionado federal, el cual es designado por el Presidente de la Republica a propuesta del Secretario de Salud, siendo la Secretaría de Salud quien supervisa a la COFEPRIS.

1.3 ESTRUCTURA.

Actualmente la Comisión se encuentra integrada por las siguientes unidades administrativas:

- **Comisión de Evidencia y Manejo de Riesgos.**
  Cuyo objetivo es proponer y conducir la aplicación de los instrumentos no regulatorios e innovadores que permitan generar evidencias y presentar alternativas para el manejo de riesgos sanitarios.

- **Comisión de Fomento Sanitario.**
  Su objetivo es coadyuvar a fortalecer la política de manejo no regulatorio así como prevenir la mejora regulatoria y el desarrollo de acciones tendientes a promover la mejora continua de la calidad de la salud de la población mediante esquemas de comunicación, capacitación, coordinación y concertación con los sectores público, privado y social.

- **Comisión de Autorización Sanitaria.**
  Su objetivo es definir los requisitos y las disposiciones administrativas para la operación de establecimientos dedicados al proceso de las materias a que se refiere el artículo 3, fracción I del Reglamento de la Comisión; así como expedir, prorrogar o revocar dichas autorizaciones sanitarias.

- **Comisión de Operación Sanitaria.**
  Evaluar el cumplimiento de las disposiciones establecidas a través de la vigilancia, supervisión y el dictamen para identificar y prevenir los riesgos sanitarios ocasionados por el uso o consumo de productos, sustancias, insumos, tecnologías y servicios, exposición a agentes nocivos, así como establecer las medidas de seguridad en caso de detección de desviaciones a los ordenamientos legales y dar seguimiento a los procedimientos administrativos de su competencia e imponer las sanciones correspondientes a los infractores de la legislación sanitaria. Asimismo, establecer mecanismos de ayuda en el territorio nacional en caso de emergencias sanitarias que afecten la salud de la población; dar seguimiento a los programas especiales; coordinar y realizar las visitas de verificación en el extranjero (“in situ”); establecer, en coordinación con las áreas competentes de la Comisión Federal, acuerdos de reconocimiento de sistemas de vigilancia sanitaria con otros países; y establecer las políticas para el desarrollo

de instrucciones de trabajo que coadyuven a las actividades de las diferentes áreas de la Comisión de Operación Sanitaria.

- **Comisión de Control Analítico y Ampliación de Cobertura.**
  Colaborar en la realización de ensayos de laboratorio a todos aquellos productos sujetos al control sanitario, emitiendo resultados confiables y oportunos para la toma de decisiones y ampliar la cobertura en las actividades de vigilancia sanitaria a través de terceros autorizados y de la Red Nacional de Laboratorios Estatales de Salud Pública, para contribuir en la prevención y protección contra riesgos sanitarios.

- **Coordinación General del Sistema Federal Sanitario.**
  Coordinar la elaboración de programas de acción y proyectos del Sistema Federal en base a las prioridades que se hayan consensuado entre las autoridades estatales y la Comisión Federal, buscando la mejora continua a través de mecanismos de evaluación. Asimismo, el coordinar las actividades de la Comisión, en su papel rector, en asuntos internacionales relacionados con el ámbito de su competencia; y administrar la infraestructura de comunicaciones, cómputo y sistemas del Sistema Federal Sanitario.

- **Coordinación Jurídica y Consultiva.**
  Autorizar y determinar el ejercicio de la defensa y representación jurídica de los actos de autoridad de la Comisión Federal; actuar como órgano de consulta jurídica de las unidades administrativas en las materias vinculadas con el ejercicio de sus atribuciones, así como participar en la creación de proyectos legislativos y demás disposiciones legales y administrativas que requieran la participación de la Comisión Federal.

- **Secretaría General**
  Asesorar y conducir con eficiencia los Recursos Humanos, Materiales, Financieros y Tecnológicos de la Comisión, apoyando a las Unidades Administrativas con metodología organizacional de Sistemas y de procedimientos, que les ayude a simplificar, automatizar y modernizar sus procesos administrativos, asimismo, proponer los mecanismos, indicadores y métodos de coordinación que les permitan cumplir en los programas y trámites.

2. PROBLEMA.

2.1 CARACTERÍSTICAS DEL PROBLEMA.

Actualmente la COFEPRIS cuenta con un sistema de administración de trámites el cual está desarrollado en CLIPPER, este sistema se encarga de la administración de trámites que atiende la Comisión desde su ingreso, su resolución y la entrega al usuario final.

Las desventajas de esta aplicación son las siguientes:

- Problemas para conjuntar la información a nivel nacional, ya que cada entidad federativa trabaja con bases de datos independientes.
- Tiempos de atención altos, ya que para tratar de resolver algún problema en una entidad federativa inicialmente debemos de compartir la información y recrear el ambiente con su información.
- Para realizar cualquier cambio en las funcionalidades del sistema se requiere programar por un experto y el tiempo puede ser extenso.
- Los usuarios no reciben alertas y alarmas.
- La explotación de los datos es difícil para los usuarios que la requieren
- No existe flexibilidad para definir y adaptar los perfiles o roles a los grupos de usuarios.
- Información fragmentada que no permite un análisis integral de riesgos y apoyo a la operación de la COFEPRIS.
- No se cuenta con comunicación en red interna y externa.
- No se tiene un modelo de un sistema unificado de información.

Por estas desventajas la COFEPRIS requiere de un nuevo sistema que ayude a centralizar la información en tiempo real para tomar decisiones de una menara mas rápida, asimismo para que la misma área de sistemas dé un servicio más eficiente a las entidades federativas al tener la información reunida en un mismo lugar, también se requiere que el sistema pueda ser accesado a través de la misma intranet de la Comisión la cual está comunicada con todas las entidades de la República, pero también por internet, ya que existen puntos dentro del país en donde no ha sido posible extender la intranet.

3. ANÁLISIS Y DIAGNÓSTICO DEL PROBLEMA.

3.1 ANÁLISIS.

La Comisión Federal para la Protección contra Riesgos Sanitarios (COFEPRIS) desarrollará y pondrá en operación un sistema integral que le permita llevar un control y monitoreo de brotes de enfermedades y epidemias relacionadas con el consumo de productos y servicios, realizar la captura de mediciones de riesgos sanitarios y monitorear las acciones realizadas para la prevención y protección a nivel nacional, estatal, jurisdiccional y municipal. El alcance de este módulo es mantener el control de los trámites que atiende la Comisión, así como la generación del padrón de establecimientos para su monitoreo y vigilancia.

El sistema a desarrollar debe interactuar con otros sistemas, así como proporcionar datos, por ejemplo:

- Información para publicar en Internet de conformidad con la Ley de Acceso a la Información (permisos otorgados, registros, licencias, avisos)
- Enlace con Tramitanet
- Acceso desde Intranet e Internet
- Conexión con otros sistemas de información:
    - Aduanas, sistema para compartir información de los permisos de importación otorgados en COFEPRIS y que son finalmente revisados en las aduanas del país.
    - NDS (National Droug System), sistema creado por la Organización de las Naciones Unidas para tener un control eficiente sobre estupefacientes y psicotrópicos.
    - PREQUIM (Precursores Químicos), sistema en donde los laboratorios reportan que sustancias químicas importaran a lo largo del año.
    - SITER (Sistema de Transferencia Electrónica de Registros), sistema mediante el cual los laboratorios proporcionan la información de los medicamentos para los cuales desean obtener un registro para comercializarlo en el país.
    - SIILAB (Sistema Integral de Información de Laboratorio), sistema mediante el cual el laboratorio nacional registra los resultados de las muestras tomadas de alimentos, medicamentos, etc.
    - Hoja de Ayuda para el pago de derechos, sistema para generar el pago de derechos sobre los trámites que atiene la COFEPRIS.
    - SISCOVAC (Sistema de Control de Vacunas), sistema con el cual se lleva un control de productos biológicos que sirven para fabricar vacunas.

3.2 OBJETIVO.

Desarrollar un módulo para la captura y emisión de resoluciones de los trámites que atiende la COFEPRIS, el cual tendrá como primer objetivo la conformación del padrón de establecimientos a nivel nacional los cuales pueden llegar a generar algún riesgo sanitario para la población

El sistema se desarrollará en base a la filosofía de trabajo de la institución por lo que se mantendrá un flujo de trabajo general para todos los trámites. El trámite debe ingresar a través del CIS, éste turna al área correspondiente para su resolución, al área regresa el trámite al CIS y éste lo entrega al usuario.

3.3 DIAGNÓSTICO

Por las características del desarrollo, en este caso el acceso a través de internet y de intranet se propone que sea un sistema en web, con lenguaje de programación java y con el framework struts, con este último separamos la lógica de negocio con la capa de presentación, asimismo hace más flexible el mantenimiento de la aplicación.

Como Base de Datos se utilizará Microsoft SQL Server 2000, ya que es un Sistema Manejador de Bases de Datos relacionales con un amplio soporte técnico, además de que en la Comisión ya se cuenta con la licencia correspondiente para la utilización del software.

Como servidor de aplicación utilizaremos el Apache Tomcat en su versión 6.0, Apache es el servidor web más empleado en el mundo y Tomcat nos da soporte como contenedor de componentes web de java como son jsp`s y servlets, además cabe mencionarlo que es software libre, pero con un extenso soporte.

4. TECNOLOGÍA JAVA Y EL FRAMEWORK STRUTS.

4.1 HISTORIA DE JAVA.

Sun Microsystems, líder en servidores para Internet, uno de cuyos lemas desde hace mucho tiempo es "the network is the computer" (lo que quiere dar a entender que el verdadero ordenador es la red en su conjunto y no cada máquina individual), es quien ha desarrollado el lenguaje Java, en un intento de resolver simultáneamente todos los problemas que se le plantean a los desarrolladores de software por la proliferación de arquitecturas incompatibles, tanto entre las diferentes máquinas como entre los diversos sistemas operativos y sistemas de ventanas que funcionaban sobre una misma máquina, añadiendo la dificultad de crear aplicaciones distribuidas en una red como Internet.

Hace algunos años, Sun Microsystems decidió intentar introducirse en el mercado de la electrónica de consumo y desarrollar programas para pequeños dispositivos electrónicos. Tras unos comienzos dudosos, Sun decidió crear una filial, denominada FirstPerson Inc., para dar margen de maniobra al equipo responsable del proyecto.

El mercado inicialmente previsto para los programas de FirstPerson eran los equipos domésticos: microondas, tostadoras y, fundamentalmente, televisión interactiva. Este mercado, dada la falta de pericia de los usuarios para el manejo de estos dispositivos, requería unos interfaces mucho más cómodos e intuitivos que los sistemas de ventanas que proliferaban en el momento.

Otros requisitos importantes a tener en cuenta eran la fiabilidad del código y la facilidad de desarrollo. James Gosling, el miembro del equipo con más experiencia en lenguajes de programación, decidió que las ventajas aportadas por la eficiencia de C++ no compensaban el gran coste de pruebas y depuración. Gosling había estado trabajando en su tiempo libre en un lenguaje de programación que él había llamado Oak, el cual, aún partiendo de la sintaxis de C++, intentaba remediar las deficiencias que iba observando.

Los lenguajes al uso, como C o C++, deben ser compilados para un chip, y si se cambia el chip, todo el software debe compilarse de nuevo. Esto encarece mucho los desarrollos y el problema es especialmente acusado en el campo de la electrónica de consumo. La aparición de un chip más barato y, generalmente, más eficiente, conduce inmediatamente a los fabricantes a incluirlo en las nuevas series de sus cadenas de producción, por pequeña que sea la diferencia en precio ya que, multiplicada por la tirada masiva de los aparatos, supone un ahorro considerable. Por tanto, Gosling decidió mejorar las características de Oak y utilizarlo.

El primer proyecto en que se aplicó este lenguaje recibió el nombre de proyecto Green y consistía en un sistema de control completo de los aparatos electrónicos y el entorno de un hogar. Para ello se construyó un ordenador experimental denominado *7 (Star Seven). El sistema presentaba una interfaz basada en la representación de la casa de forma animada y el control se llevaba a cabo mediante una pantalla sensible al tacto. En el sistema aparecía Duke, la actual mascota de Java.

Posteriormente se aplicó a otro proyecto denominado VOD (Video On Demand) en el que se empleaba como interfaz para la televisión interactiva. Ninguno de estos proyectos se convirtió nunca en un sistema comercial, pero fueron desarrollados enteramente en un Java primitivo y fueron como su bautismo de fuego.

Una vez que en Sun se dieron cuenta de que a corto plazo la televisión interactiva no iba a ser un gran éxito, urgieron a FirstPerson a desarrollar con rapidez nuevas estrategias que produjeran beneficios. No lo consiguieron y FirstPerson cerró en la primavera de 1994.

Pese a lo que parecía ya un olvido definitivo, Bill Joy, cofundador de Sun y uno de los desarrolladores principales del Unix de Berkeley, juzgó que Internet podría llegar a ser el campo de juego adecuado para disputar a Microsoft su primacía casi absoluta en el terreno del software, y vio en Oak el instrumento idóneo para llevar a cabo estos planes.

Sun anuncio formalmente Java en una conferencia importante en mayo de 1995. Ordinariamente un evento de tal naturaleza no habría generado tanta atención. Sin embargo, Java hizo surgir un interés inmediato en la comunidad de los negocios en vista de la magnitud del interés comercial por la World Wide Web. Java no era un lenguaje académico como Pascal o un lenguaje diseñado por una sola persona o un grupo reducido para su propio uso local como C o C++. Más bien, Java había sido diseñado con motivos comerciales y genero un interés avasallador en la comunidad de los negocios a causa de otro avance relacionado con la Internet, la World Wide Web. [1]

4.2 CARACTERÍSTICAS DE JAVA.

Las características principales que nos ofrece Java respecto a cualquier otro lenguaje de programación son:

Simple

Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero C y C++ son lenguajes más difundidos, por ello Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje.

Java elimina muchas de las características de otros lenguajes como C++, para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el garbage collector (reciclador de memoria dinámica). No es necesario preocuparse de liberar memoria, el reciclador se encarga de ello y como es un thread de baja prioridad, cuando entra en acción, permite liberar bloques de memoria muy grandes, lo que reduce la fragmentación de la memoria.

[1] Deitel, H.M y Deitel, P.J (1998) *Como Programar en Java*, Primera Edición: Prentince Hall

Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos, entre las que destacan:

aritmética de punteros
no existen referencias
registros (struct)
definición de tipos (typedef)
macros (#define)
necesidad de liberar memoria (free)

Aunque, en realidad, lo que hace es eliminar las palabras reservadas (struct, typedef), ya que las clases son algo parecido.

Además, el intérprete completo de Java que hay en este momento es muy pequeño, solamente ocupa 215 Kb de RAM.

Orientado a objetos

Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Las plantillas de objetos son llamadas, como en C++, clases y sus copias, instancias. Estas instancias, como en C++, necesitan ser construidas y destruidas en espacios de memoria.

Java incorpora funcionalidades inexistentes en C++ como por ejemplo, la resolución dinámica de métodos. Esta característica deriva del lenguaje Objective C, propietario del sistema operativo Next. En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (RunTime Type Identification) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases en Java tienen una representación en el runtime que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda.

Distribuido

Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

Robusto

Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria.

También implementa los arrays auténticos, en vez de listas enlazadas de puntero con comprobación de límites, para evitar la posibilidad de sobrescribir o corromper memoria, resultado de punteros que señalan a zonas equivocadas. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java.

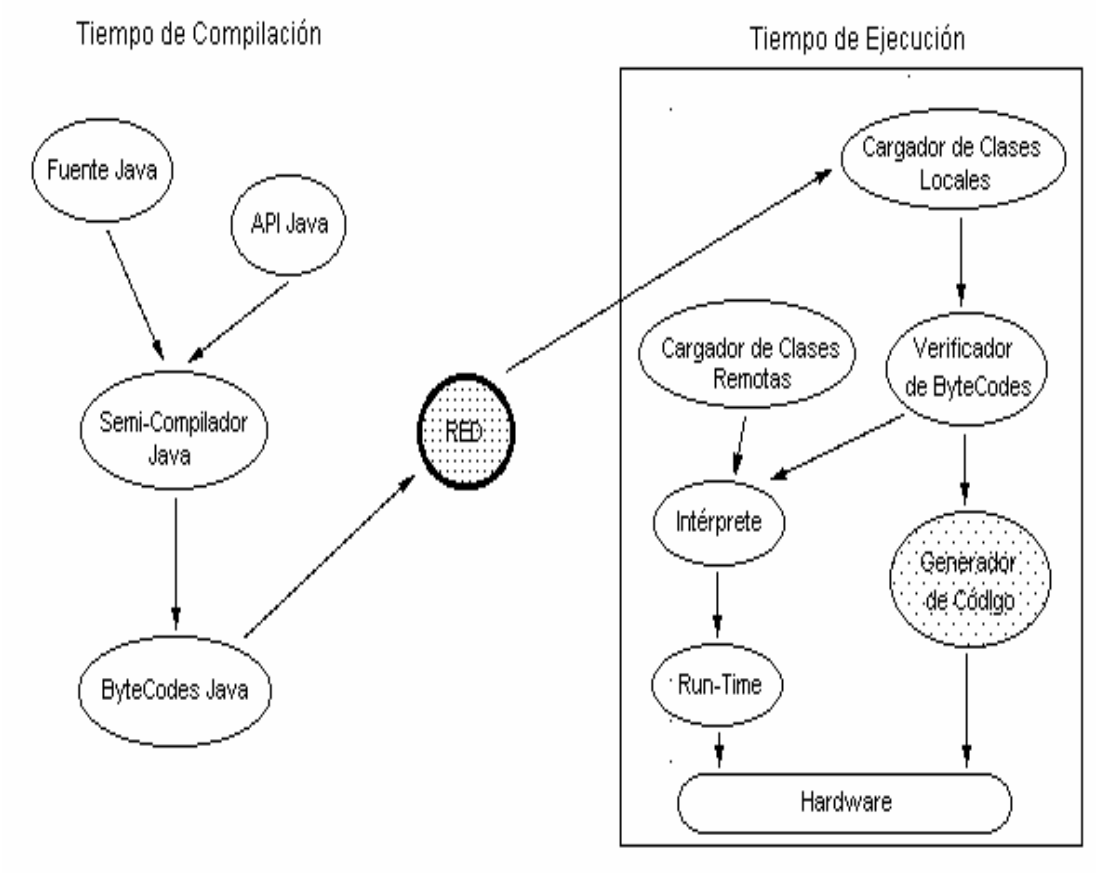
Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los byte-codes, que son el resultado de la compilación de un programa Java. Es un código de máquina virtual que es interpretado por el intérprete Java. No es el código máquina directamente entendible por el hardware, pero ya ha pasado todas las fases del compilador: análisis de instrucciones, orden de operadores, etc., y ya tiene generada la pila de ejecución de órdenes.

Java proporciona:

Comprobación de punteros
Comprobación de límites de arrays
Excepciones
Verificación de byte-codes

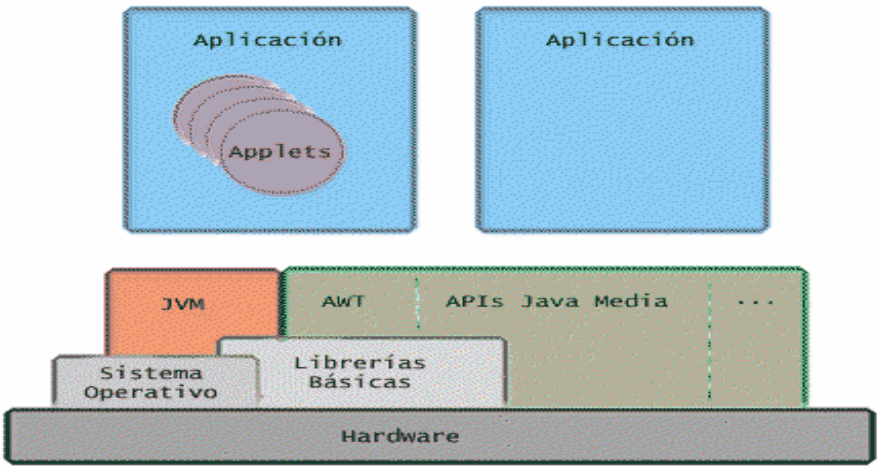
Arquitectura neutral

Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado. Actualmente existen sistemas run-time para Solaris 2.x, SunOs 4.1.x, Windows 95, Windows NT, Linux, Irix, Aix, Mac, Apple y probablemente haya grupos de desarrollo trabajando en el porting a otras plataformas.



El código fuente Java se "compila" a un código de bytes de alto nivel independiente de la máquina. Este código (byte-codes) está diseñado para ejecutarse en una máquina hipotética que es implementada por un sistema run-time, que sí es dependiente de la máquina.

En una representación en que tuviésemos que indicar todos los elementos que forman parte de la arquitectura de Java sobre una plataforma genérica, obtendríamos una figura como la siguiente:



En ella podemos ver que lo verdaderamente dependiente del sistema es la Máquina Virtual Java (JVM) y las librerías fundamentales, que también nos permitirían acceder directamente al hardware de la máquina. Además, habrá APIs de Java que también entren en contacto directo con el hardware y serán dependientes de la máquina, como ejemplo de este tipo de APIs podemos citar:

Java 2D: gráficos 2D y manipulación de imágenes
Java Media Framework: Elementos críticos en el tiempo: audio, video...
Java Animation: Animación de objetos en 2D
Java Telephony: Integración con telefonía
Java Share: Interacción entre aplicaciones multiusuario
Java 3D: Gráficos 3D y su manipulación
Entre muchas otras.

Seguro

La seguridad en Java tiene dos facetas. En el lenguaje, las características como los punteros o el casting implícito que hacen los compiladores de C y C++ se eliminan para prevenir el acceso ilegal a la memoria. Cuando se usa Java para crear un navegador, se combinan las características del lenguaje con protecciones de sentido común aplicadas al propio navegador.

El lenguaje C, por ejemplo, tiene lagunas de seguridad importantes, como son los errores de alineación. Los programadores de C utilizan punteros en conjunción con operaciones aritméticas. Esto le permite al programador que un puntero reverencié a un lugar conocido de la memoria y pueda sumar (o restar) algún valor, para referirse a otro lugar de la memoria. Si otros programadores conocen nuestras estructuras de datos pueden extraer información confidencial de nuestro sistema. Con un lenguaje como C, se pueden tomar números enteros aleatorios y convertirlos en punteros para luego acceder a la memoria:

```
printf( "Escribe un valor entero: " );
scanf( "%u",&puntero );
printf( "Cadena de memoria: %s\n",puntero );
```

Otra laguna de seguridad u otro tipo de ataque es el Caballo de Troya. Se presenta un programa como una utilidad, resultando tener una funcionalidad destructiva. Por ejemplo, en UNIX se visualiza el contenido de un directorio con el comando ls. Si un programador deja un comando destructivo bajo esta referencia, se puede correr el riesgo de ejecutar código malicioso, aunque el comando siga haciendo la funcionalidad que se le supone, después de lanzar su carga destructiva.

Por ejemplo, después de que el caballo de Troya haya enviado por correo el /etc/shadow a su creador, ejecuta la funcionalidad de ls presentando el contenido del directorio. Se notará un retardo, pero nada inusual.

El código Java pasa muchos tests antes de ejecutarse en una máquina. El código se pasa a través de un verificador de byte-codes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal

(código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto).

Si los byte-codes pasan la verificación sin generar ningún mensaje de error, entonces sabemos que:

El código no produce desbordamiento de operandos en la pila. El tipo de los parámetros de todos los códigos de operación son conocidos y correctos. No ha ocurrido ninguna conversión ilegal de datos, tal como convertir enteros en punteros. El acceso a los campos de un objeto se sabe que es legal: public, private, protected. No hay ningún intento de violar las reglas de acceso y seguridad establecidas

El Cargador de Clases también ayuda a Java a mantener su seguridad, separando el espacio de nombres del sistema de ficheros local, del de los recursos procedentes de la red. Esto limita cualquier aplicación del tipo Caballo de Troya, ya que las clases se buscan primero entre las locales y luego entre las procedentes del exterior.

Las clases importadas de la red se almacenan en un espacio de nombres privado, asociado con el origen. Cuando una clase del espacio de nombres privado accede a otra clase, primero se busca en las clases predefinidas (del sistema local) y luego en el espacio de nombres de la clase que hace la referencia. Esto imposibilita que una clase suplante a una predefinida.

En resumen, las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus. Java no posee una semántica específica para modificar la pila de programa, la memoria libre o utilizar objetos y métodos de un programa sin los privilegios del kernel del sistema operativo. Además, para evitar modificaciones por parte de los crackers de la red, implementa un método ultraseguro de autentificación por clave pública. El Cargador de Clases puede verificar una firma digital antes de realizar una instancia de un objeto. Por tanto, ningún objeto se crea y almacena en memoria sin que se validen los privilegios de acceso. Es decir, la seguridad se integra en el momento de compilación, con el nivel de detalle y de privilegio que sea necesario.

Dada la concepción del lenguaje y si todos los elementos se mantienen dentro del estándar marcado por Sun, no hay peligro. Java imposibilita, también, abrir ningún fichero de la máquina local (siempre que se realizan operaciones con archivos, éstas trabajan sobre el disco duro de la máquina de donde partió el applet), no permite ejecutar ninguna aplicación nativa de una plataforma e impide que se utilicen otros ordenadores como puente, es decir, nadie puede utilizar nuestra máquina para hacer peticiones o realizar operaciones con otra. Además, los intérpretes que incorporan los navegadores de la Web son aún más restrictivos. Bajo estas condiciones (y dentro de la filosofía de que el único ordenador seguro es el que está apagado, desenchufado, dentro de una cámara acorazada en un bunker y rodeado por mil soldados de los cuerpos especiales del ejército), se puede considerar que Java es un lenguaje seguro y que los applets están libres de virus.

Respecto a la seguridad del código fuente, no ya del lenguaje, JDK proporciona un desensamblador de byte-code, que permite que cualquier programa pueda ser convertido a código fuente, lo que para el programador significa una vulnerabilidad total a su

código. Utilizando javap no se obtiene el código fuente original, pero sí desmonta el programa mostrando el algoritmo que se utiliza, que es lo realmente interesante. La protección de los programadores ante esto es utilizar llamadas a programas nativos, externos (incluso en C o C++) de forma que no sea descompilable todo el código; aunque así se pierda portabilidad. Esta es otra de las cuestiones que Java tiene pendientes.

Portable

Más allá de la portabilidad básica por ser de arquitectura independiente, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Los enteros son siempre enteros y además, enteros de 32 bits en complemento a 2. Además, Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas de forma que las ventanas puedan ser implantadas en entornos Unix, Pc o Mac.

Interpretado

El intérprete Java (sistema run-time) puede ejecutar directamente el código objeto. Enlazar (linkar) un programa, normalmente, consume menos recursos que compilarlo, por lo que los desarrolladores con Java pasarán más tiempo desarrollando y menos esperando por el ordenador. No obstante, el compilador actual del JDK es bastante lento. Por ahora, que todavía no hay compiladores específicos de Java para las diversas plataformas, Java es más lento que otros lenguajes de programación, como C++, ya que debe ser interpretado y no ejecutado como sucede en cualquier programa tradicional.

Se dice que Java es de 10 a 30 veces más lento que C, y que tampoco existen en Java proyectos de gran envergadura como en otros lenguajes. La verdad es que ya hay comparaciones ventajosas entre Java y el resto de los lenguajes de programación, y una ingente cantidad de folletos electrónicos que supuran fanatismo en favor y en contra de los distintos lenguajes contendientes con Java. Lo que se suele dejar de lado en todo esto, es que primero habría que decidir hasta que punto Java, un lenguaje en pleno desarrollo y todavía sin definición definitiva, está maduro como lenguaje de programación para ser comparado con otros; como por ejemplo con Smalltalk, que lleva más de 20 años en cancha.

La verdad es que, Java para conseguir ser un lenguaje independiente del sistema operativo y del procesador que incorpore la máquina utilizada, es tanto interpretado como compilado. Y esto no es ningún contrasentido, me explico, el código fuente escrito con cualquier editor se compila generando el byte-code. Este código intermedio es de muy bajo nivel, pero sin alcanzar las instrucciones máquina propias de cada plataforma y no tiene nada que ver con el p-code de Visual Basic. El byte-code corresponde al 80% de las instrucciones de la aplicación. Ese mismo código es el que se puede ejecutar sobre cualquier plataforma. Para ello hace falta el run-time, que sí es completamente dependiente de la máquina y del sistema operativo, que interpreta dinámicamente el byte-code y añade el 20% de instrucciones que faltaban para su ejecución. Con este sistema es fácil crear aplicaciones multiplataforma, pero para ejecutarlas es necesario que exista el run-time correspondiente al sistema operativo utilizado.

Multithreaded

Al ser multithreaded (multihilvanado, en mala traducción), Java permite muchas actividades simultáneas en un programa. Los threads (a veces llamados, procesos ligeros), son básicamente pequeños procesos o piezas independientes de un gran proceso. Al estar los threads contruidos en el lenguaje, son más fáciles de usar y más robustos que sus homólogos en C o C++.

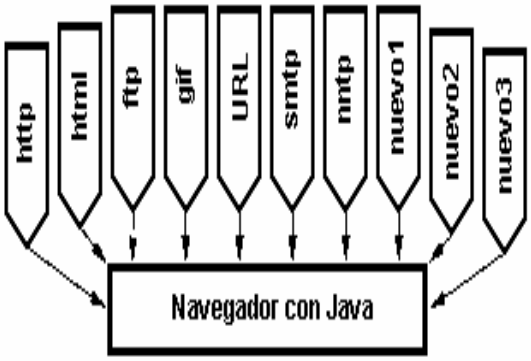
El beneficio de ser miltithreaded consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), aún supera a los entornos de flujo único de programa (single-threaded), tanto en facilidad de desarrollo como en rendimiento.

Cualquiera que haya utilizado la tecnología de navegación concurrente, sabe lo frustrante que puede ser esperar por una gran imagen que se está trayendo. En Java, las imágenes se pueden ir trayendo en un thread independiente, permitiendo que el usuario pueda acceder a la información en la página sin tener que esperar por el navegador.

Dinámico

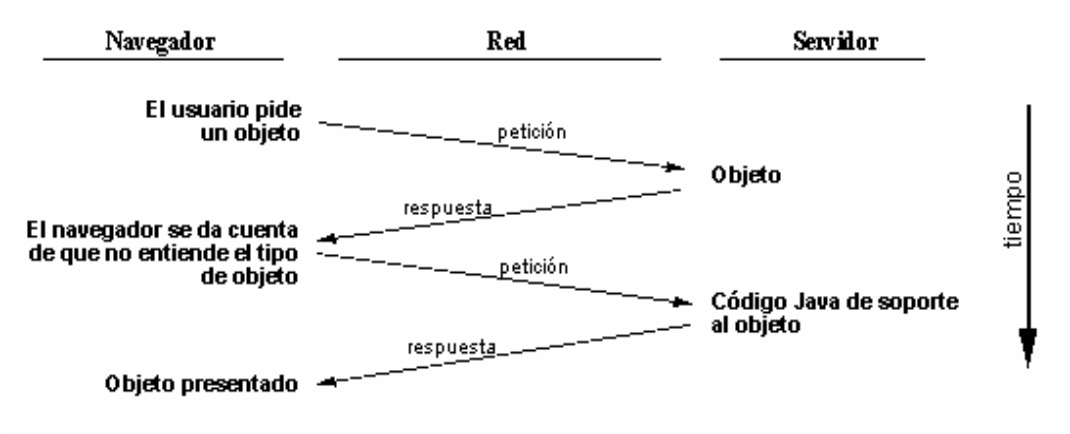
Java se beneficia todo lo posible de la tecnología orientada a objetos. Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución. Las librerías nuevas o actualizadas no paralizarán las aplicaciones actuales (siempre que mantengan el API anterior). Como lo muestra la siguiente figura:

Monolito: cada pieza de código se compacta dentro del código del navegador



Sistema Federado: el navegador es un coordinador de piezas, y cada pieza es responsable de una función. Las piezas se pueden añadir dinámicamente a través de la red

Java también simplifica el uso de protocolos nuevos o actualizados. Si su sistema ejecuta una aplicación Java sobre la red y encuentra una pieza de la aplicación que no sabe manejar, tal como se ha explicado en párrafos anteriores, Java es capaz de traer automáticamente cualquiera de esas piezas que el sistema necesita para funcionar, como se muestra en la siguiente figura:



Java, para evitar que los módulos de byte-codes o los objetos o nuevas clases, haya que estar trayéndolos de la red cada vez que se necesiten, implementa las opciones de persistencia, para que no se eliminen cuando de limpie la caché de la máquina.

¿Cuál es la ventaja de todo esto? ¿Qué gano con Java?

Primero: No debes volver a escribir el código si quieres ejecutar el programa en otra máquina. Un solo código funciona para todos los browsers compatibles con Java o donde se tenga una Máquina Virtual de Java (Mac's, PC's, Sun's, etc).
Segundo: Java es un lenguaje de programación orientado a objetos, y tiene todos los beneficios que ofrece esta metodología de programacion (más adelante doy una pequeña introducción a la filosofía de objetos).
Tercero: Un browser compatible con Java deberá ejecutar cualquier programa hecho en Java, esto ahorra a los usuarios tener que estar insertando "plug-ins" y demás programas que a veces nos quitan tiempo y espacio en disco.
Cuarto: Java es un lenguaje y por lo tanto puede hacer todas las cosas que puede hacer un lenguaje de programación: Cálculos matemáticos, procesadores de palabras, bases de datos, aplicaciones gráficas, animaciones, sonido, hojas de cálculo, etc.
Quinto: Si lo que me interesa son las páginas de Web, ya no tienen que ser estáticas, se le pueden poner toda clase de elementos multimedia y permiten un alto nivel de interactividad, sin tener que gastar en paquetes carísimos de multimedia.

Todo esto suena muy bonito pero también se tienen algunas limitantes:

La velocidad. Los programas hechos en Java no tienden a ser muy rápidos, supuestamente se está trabajando en mejorar esto. Como los programas de Java son interpretados nunca alcanzan la velocidad de un verdadero ejecutable.

Java es un lenguaje de programación. Esta es otra gran limitante, por más que digan que es orientado a objetos y que es muy fácil de aprender sigue siendo un lenguaje y, por lo tanto, aprenderlo no es cosa fácil. Especialmente para los no programadores,
Java es nuevo. En pocas palabras todavía no se conocen bien todas sus capacidades.

4.3 MVC EL MODELO VISTA CONTROLADOR.

Los patrones de diseño (del inglés Design Patterns) son modelos de trabajo enfocados a dividir un problema en partes de modo que nos sea posible abordar cada una de ellas por separado para simplificar su resolución. Una posible definición de patrón de diseño sería la siguiente:

Un patrón de diseño es un conjunto de reglas que describen como afrontar tareas y solucionar problemas que surgen durante el desarrollo de software.

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

El patrón fue descrito por primera vez en 1979 por Trygve Reenskaug, entonces trabajando en Smalltalk en laboratorios de investigación de Xerox. La implementación original está descrita a fondo en Programación de Aplicaciones en Smalltalk-80(TM): Como utilizar Modelo Vista Controlador. [2]

Tenemos la siguiente descripción de los elementos que intervienen en el MVC.

Modelo: Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de éstos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o importes en un carrito de la compra.

Vista: Éste presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

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

Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos. En MVC corresponde al modelo. Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace)

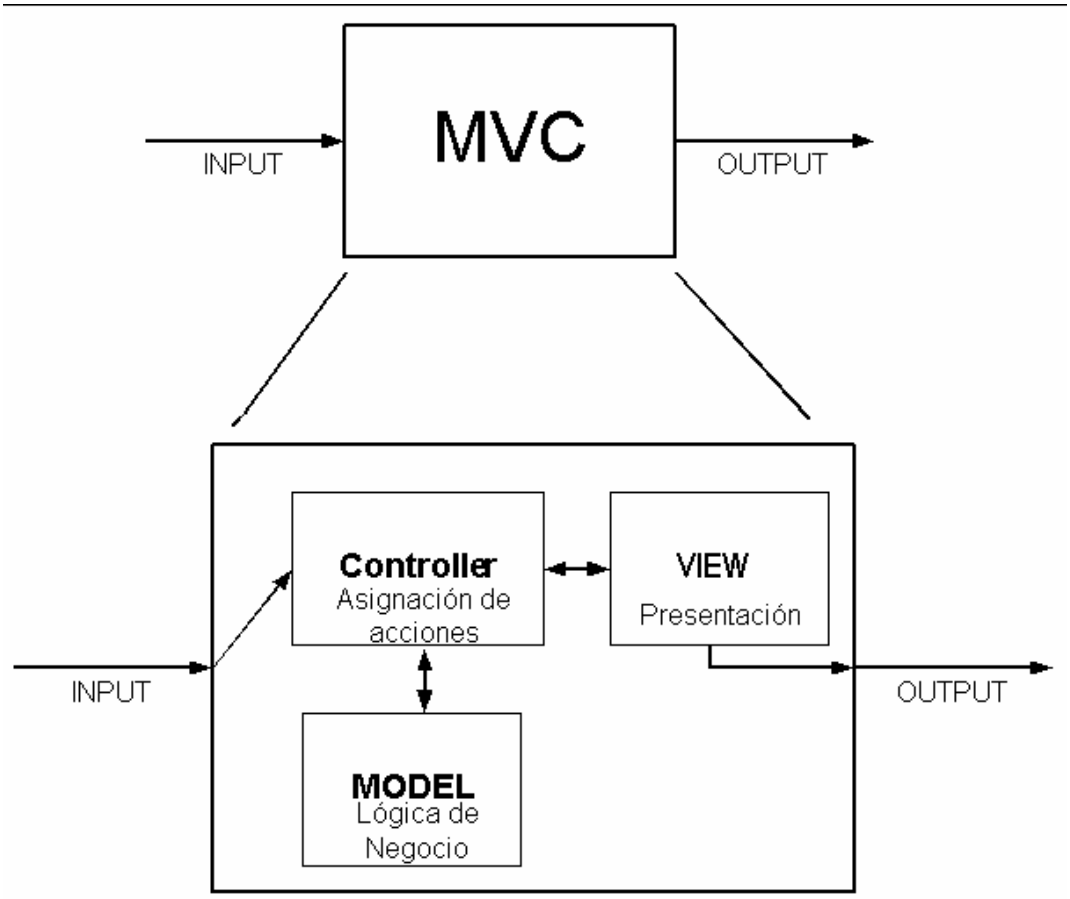
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

[2] http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html

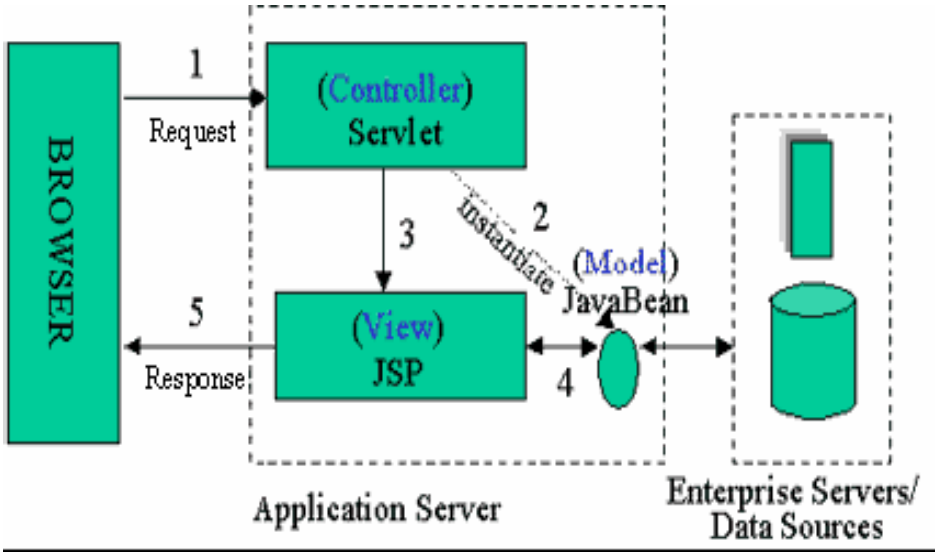
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente. Como se muestra en la siguiente figura.



La mayoría, por no decir todos, de los frameworks para Web implementan este patrón. Una aplicación de este patrón en entornos Java para programación Web es lo que se conoce con el nombre de arquitectura model 2.



Esta arquitectura consiste, a grandes rasgos, en la utilización de servlets para procesar las peticiones (controladores) y páginas JSP para mostrar la interfaz de usuario (vistas), implementando la parte del modelo mediante JavaBeans o POJOs (Plain Old Java Object) que son clases java puras sin ninguna asociación.

4.4 EL FRAMEWORK STRUTS.

El concepto framework se emplea en muchos ámbitos del desarrollo de sistemas de software, no solo en el ámbito de aplicaciones Web. Podemos encontrar frameworks para el desarrollo de aplicaciones médicas, de visión por computador, para el desarrollo de juegos, y para cualquier ámbito se nos pueda ocurrir.

En general, con el término framework, nos estamos refiriendo a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta.

Struts inicialmente pertenecía al proyecto Jakarta de la fundación Apache, pero se independizó y pasó a conocerse como Apache Struts. Básicamente, se presenta como un framework de soporte de desarrollo de aplicaciones web siguiendo el conocido patrón MVC (Modelo-Vista-Controlador) corriendo bajo J2EE. Este patrón arquitectónico separa los datos de una aplicación software, su interfaz de usuario y la lógica de cada uno de las entidades conceptuales que la forman, en tres componentes distintos y es muy extendido y recomendable su uso en el desarrollo de aplicaciones web, siendo uno

de los más fuertes pilares sobre los que se sustentan actualmente las buenas prácticas de diseño de aplicaciones software para la web donde la vista se centra al HTML de la página contenedora del servicio.

Bajo esta idea, Struts se presenta como un enorme aliado a la hora de construir aplicaciones web de una forma elegante y con un diseño basado en una de las corrientes más influyentes dentro de las buenas prácticas del desarrollo web.

El desarrollo de una aplicación web mediante Struts se puede subdividir en varias tareas, en primer lugar tendremos que definir la lógica de control de una entidad, persistente o no, que necesitemos para implementar la funcionalidad del servicio o aplicación que estemos desarrollando. Por ejemplo, supongamos que deseamos desarrollar un sistema de e-venta de entradas o tickets para determinados eventos culturales. Después de un proceso de análisis del software podremos ser capaces de identificar que necesitamos una serie de objetos o entidades en nuestra aplicación capaces de representar los conceptos que manejamos en ese dominio, como pueden ser una entrada, un evento o el mismo usuario de la web que actuará de comprador, y que estos a su vez, deberán almacenar la información necesaria para representar eficazmente a cada uno de ellos.

Una vez implementados estos elementos como clases Java con sus correspondientes constructores, y métodos de acceso y consulta a los atributos de estas, los conocidos como getters y setters, necesitaremos mantener una relación con la base de datos que soportará el almacenamiento de cada uno de los registros que se adecuen a esas entidades persistentes. Por lo que generaremos, si es necesario, las correspondientes clases para tal fin, una buena práctica es hacer uso de patrones DAO. El cómo la arquitectura apunta a un determinado JDBC para acceder a una base de datos se lleva a cabo mediante la configuración de un simple fichero de properties.

Con esto tendremos definida la lógica de negocio de nuestra aplicación la cual ha sido apartada totalmente del tipo de aplicación software que hará uso de ella, como podemos observar hasta ahora solo disponemos de clases Java que podremos enlazar con servicios web o con aplicaciones cliente indistintamente sin ningún problema. A continuación vamos a definir la presentación de los datos en la aplicación, para ello deberemos definir una serie de formularios contenedores de lo que necesitemos, volviendo al ejemplo si estamos creando un formulario que una vez introducidos los datos del usuario y del evento al pulsar un botón denominado “Comprar” nos genere un ticket para ese usuario y asociado a ese evento pues tendremos que definir una clase Java para este formulario que podríamos denominar VentaForm que almacenase todos los datos necesarios, en este caso una instancia de un objeto de tipo Entrada, otra de tipo Evento y otra de tipo Ticket, que no son, ni más ni menos, que las anteriormente desarrolladas.

Y por último tendremos que definir la acción asociada al evento de pulsar el citado botón, que es donde realmente se define la funcionalidad explícita en el caso de uso que estamos comentando. Esta acción se define como una clase especial denominada AccionComprar que deberá heredar de una clase del sistema denominada Action y que únicamente posee un método: execute. El método execute es aquel que modela la funcionalidad que el cliente desea para con un determinado objeto de un formulario, sea o no un formulario web, insistimos en la independencia total hasta ahora del patrón con

respecto a la tecnología. Una vez definido el método este devolverá un valor denominado actionForward, algo así como resultado de ejecución de la funcionalidad implícita en la acción a modelar. En nuestro ejemplo podríamos decir que este valor es "ok" cuando la venta se realiza de forma correcta (los datos del usuario son válidos, el evento seleccionado dispone aún de tickets a la venta, etc.) y "error" en otro caso.

Ahora es cuando Struts entra en el juego. Únicamente tendremos que definir con este soporte el plano de correlaciones entre los beans de formularios, las acciones a ejecutar y la redirección en función de los valores del actionForward, con lo que con la edición de un sencillo código en formato XML tendremos toda la estructura montada. Esta edición no tiene porque ser "a pelo" existiendo múltiples plugins para aplicaciones como Eclipse o IBM Websphere que nos introducen asistentes para una más fácil, si cabe, resolución de asociaciones. La asociación consiste en definir lo siguiente: dada esta acción (AccionComprar) que trabaja sobre el formulario (VentaForm) que contiene los datos siguientes (los relativos al evento y al usuario, así como al ticket a generar, que inicialmente será una entidad vacía), redirige el flujo de la aplicación a la localización X si el forward obtenido es "ok" o sino a la localización Y, si el forward es "error". De tal forma la arquitectura posee conocimiento de todo el flujo de posibilidades que se pueden presentar en la aplicación.

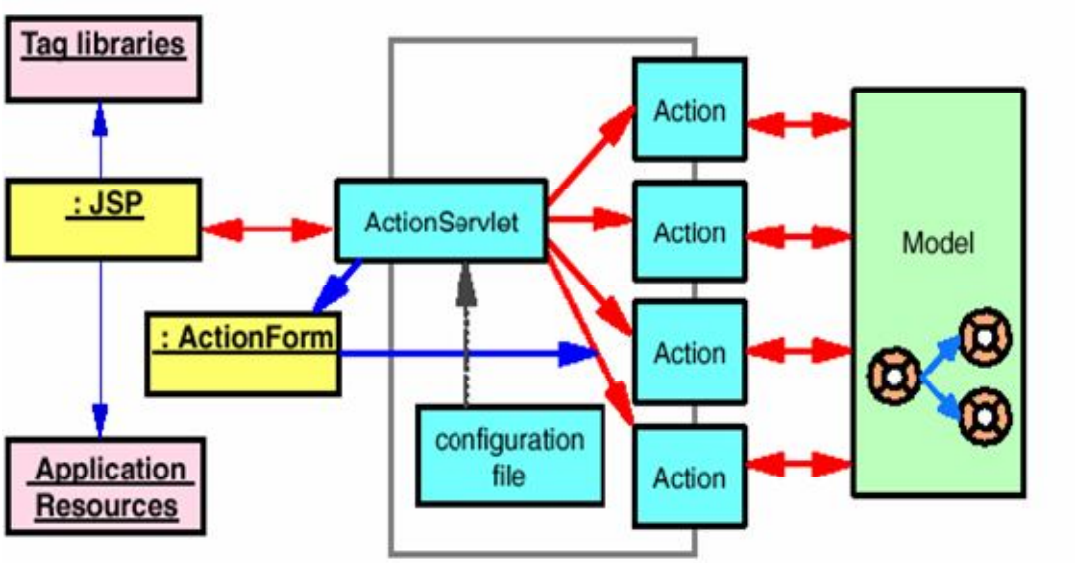
Como podemos ver aún no hemos metido mano al código HTML propio de la aplicación web, y es que es mucho más recomendable que esto el desarrollo de un etiquetado propio en JSP y montar la aplicación mediante este lenguaje de cliente, con lo que mediante el paso de parámetros por el contexto del dominio de aplicación entre la zona del cliente y del servidor será el puente entre ambos para compartir información.

Lo único que nos queda es poder definir de que forma invocar a una de las ramas de flujo definidas en el fichero XML de correlaciones de acción. Esto se llevará a cabo invocando desde un determinado evento, en este caso el evento onclick del botón "Comprar", mediante un sencillo código JavaScript la rama a invocar mediante el nombre de la acción concatenada con la extensión ".do", esta dirección será interpretada por el servidor y buscará dentro de los diversos ficheros de correlación cúal es la acción a ser ejecutada invocando Struts automáticamente el código del método execute. Así pues, vemos que la arquitectura de Struts nos permite definir toda la aplicación web de forma independiente a cómo se presentará en la web, centrándose exclusivamente en la funcionalidad o servicio a brindar.

Entre los inconvenientes a citar, la necesidad de montar un servidor de tipo Apache Tomcat para interpretar el JSP, el cual consume más recursos que un típico Apache sencillo con soporte para lenguajes como PHP.

La conclusión, un marco de trabajo idóneo para realizar buenas y consistentes aplicaciones para la web y con una expansión enorme, alta demanda en el mercado y con un competidor no menos fuerte: Spring, que hace que el ritmo de actualización de Struts y de incremento de funcionalidades nuevas para este sea constante, lo cual siempre es una buena noticia para los desarrolladores web.

La siguiente figura muestra la arquitectura de una aplicación struts.



4.4.1 Reglas para escribir apropiadamente una aplicación usando el modelo MVC con Struts.

Reglas para la capa Vista.

1. Una vista nunca seleccionará la siguiente vista
2. Tratar de usar solamente un JSP que no tenga nada de lógica
3. Mantener sus JSPs simples y el mantenimiento de su aplicación web será mucho más fácil.
4. Los JSP no deben tener mucha lógica.

Reglas para la Capa de Control.

1. Las acciones deben ser pequeñas.
2. Las acciones no deben de tener relación directamente con el JDBC ni con EJBs o Hibernate, o etc.
3. Las acciones nunca implementarán reglas de negocio o cálculos complejos.

Reglas para el Modelo

1. El modelo se debe divorciar totalmente del controlador, ya que los objetos pueden ser reemplazados por otros, con esto debemos lograr que el controlador funcione independientemente de la implementación del modelo.

5. PROPUESTA DE SOLUCIÓN.

5.1 OBJETIVO DEL PROYECTO.

Desarrollar un módulo para la captura y emisión de resoluciones de los trámites que atiende la Comisión Federal para la Protección contra Riesgos Sanitarios (COFEPRIS), el cual tendrá como primer objetivo la conformación del padrón de establecimientos a nivel nacional los cuales pueden llegar a generar algún riesgo sanitario para la población

El sistema se desarrollará en base a la filosofía de trabajo de la institución, por lo que se mantendrá un flujo de trabajo general para todos los trámites; el trámite debe ingresar a través del Centro Integral de Servicios (CIS), éste turna al área correspondiente para su resolución, el área regresa el trámite al CIS y éste lo entrega al usuario.

5.2 REQUERIMIENTOS.

5.2.1 Requerimientos Funcionales.

- El sistema deberá contar con un módulo de accesos, en donde a través de una cuenta de usuario y un password personalizado el usuario entrará al sistema y tendrá acceso a los módulos correspondientes con su perfíl.

- El sistema debe contar con un Módulo de Ingreso de Trámites, el cual será operado por personal de ventanilla (el cual es parte del Centro Integral de Servicios) que estará encargado de dar una atención rápida al usuario final para que ingrese sus trámites de una forma fácil. En este módulo se debe seleccionar el tipo de trámite que ingresa y está compuesto por Formato, Tipo de Trámite, Subtipo de Trámite y Modalidad. También en este módulo se debe contar con una opción para seleccionar la forma de entrada del trámite, es decir, si entró por ventanilla, fax, correo, etc. Dentro de este módulo debe haber validaciones para ingresar un trámite, normalmente todos los trámites requieren que sean asociados a personas morales, es decir, establecimientos y que éstos se encuentren dados de alta dentro del mismo sistema, si se trata de usuarios ocasionales o particulares, el sistema debe permitir la captura de los datos mínimos necesarios para ingresarlos dentro del padrón de establecimientos.

- El sistema debe contar con un módulo de Captura, en el cual una vez que los trámites han sido ingresados a través de las ventanillas de atención a usuarios, pasarán a un área de captura en donde con la documentación entregada por el usuario y de acuerdo al tipo de trámite se obtendrán los datos necesarios para continuar con el proceso establecido. Cabe mencionar que las pantallas de captura varían con respecto al tipo de trámite.

- El sistema debe contar con un módulo de Entrega/Recepción de trámites en donde las mesas de control de cada área recibirán y entregaran sus trámites correspondientes, en donde según el flujo establecido para cada tipo de trámite deberán grabarse las fechas, horas y personas responsables que hagan la entrega o recepción entre las distintas áreas involucradas en el flujo de trabajo. Cabe mencionar que la mayoría de los trámites deben tener un área asignada.

- El sistema debe contar con un módulo de resoluciones que será usado por las áreas de la institución que se encargaran de revisar el trámite y su documentación entregada, todo esto a través de un formato de dictamen previamente establecido para cada tipo de trámite, para finalmente emitir una resolución, la cual puede ser positiva, negativa, prevención, deshecho, etc.

- El sistema debe contar con un módulo para liberar resoluciones, es en este punto en donde el Mando Medio (Gerente, Subdirector, Director, Comisionado u homologo) autoriza la resolución asociada a un trámite, la cual es la respuesta hacia el usuario final por parte de la Comisión.

- El sistema debe contar con un módulo de Consultas, que será para los trámites ingresados y para los establecimientos registrados dentro del sistema en donde se proporcionara la información necesaria de éstos cuando así se requiera.

- El sistema debe contar con un módulo de Reportes, en donde de acuerdo al perfil del usuario, que normalmente será un Mando Medio, podrá consultar diversos listados de trámites y establecimientos, así como estadísticas que ayuden a la toma de decisiones.

- El sistema deberá contar con un módulo en donde el usuario podrá personalizar su cuenta cambiando su password por seguridad.

- El sistema deberá contar con un módulo de Administración de usuarios, en donde se crearan las cuentas para ingresar al sistema y se les asignará los diversos atributos de acuerdo al perfil del usuario.

- El sistema debe contar con un módulo de Control interno para los ciertos usuarios, los cuales en este módulo podrán realizar operaciones delicadas como la cancelación de trámites o modificación de estos mismos.

5.2.2 Requerimientos no Funcionales.

- El sistema deberá realizarse con los colores, imágenes y logotipos de acuerdo a la imagen institucional que se ha definido por parte del área de Comunicación.

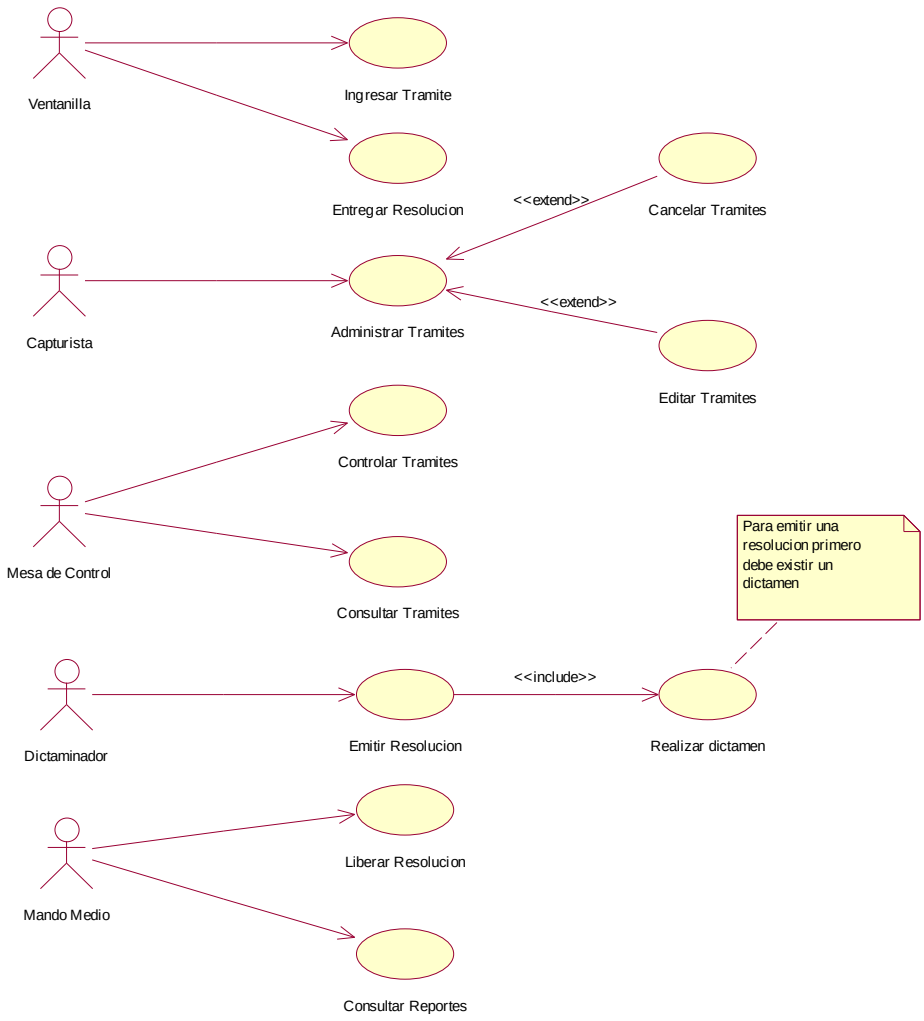
- El sistema debe estar disponible a través de la Intranet de la institución, pero también debe estar disponible a través de Internet, ya que algunas entidades de la República Mexicana no cuentan con acceso a dicha intranet.

- En cuanto a seguridad, el sistema debe manejar sesiones de usuario y no permitir que un mismo usuario se firme más de una vez.

- El sistema debe ser construido sobre la base de un desarrollo evolutivo e incremental, de manera tal que nuevas funcionalidades y requerimientos relacionados puedan ser incorporados afectando el código existente de la menor manera posible; para ello deben incorporarse aspectos de reutilización de componentes.

5.3 NARRATIVAS.

5.3.1 Diagrama de Casos de Uso General



5.3.2 Ingresar Trámite.

Generales del Caso de Uso		
Nombre Caso Uso	Ingresar Trámite	
Objetivo		
Ingresar un trámite solicitado por el usuario final.		
Nivel del Caso de Uso	**Prioridad**	**Complejidad**
Nivel 1 (Caso de Uso de Usuario)	Alta	Alta
Actores involucrados		
Ventanilla.		
Evento Disparador (Trigger)		
El usuario final (empresa o particular), ingresa un trámite a la Comisión para obtener una respuesta.		
Precondiciones		
El tipo de trámite debe estar dado de alta en el sistema para que se pueda ingresar. Si se trata de un establecimiento (persona moral), debe estar previamente dado de alta en el padrón de establecimientos.		
Post-condiciones		
Después de que se ingresa el trámite, debe de generar un acuse con el número asignado dentro del sistema para que el usuario lo use como referencia para obtener su respuesta. Una vez ingresado al sistema, el trámite se turna al capturista para que complemente su información. **EX01,** No se ingresa el trámite porque no está definido dentro del sistema		

Diagrama de Caso de Uso

Escenario Principal.

Paso	Acción
1	El usuario Ventanilla ingresa a la pantalla de ingreso de trámite.
2	El sistema le presenta la pantalla para ingresar el trámite.
3	Ventanilla selecciona el Formato, el Tipo de trámite, el Subtipo de trámite y la Modalidad para ingresar dentro del sistema el trámite.
4	El sistema valida que el trámite exista dentro del sistema. **(EX01) (RN01)**
5	Ventanilla ingresa la referencia de el usuario final, para una persona moral requiere el RFC y para un usuario particular se requiere el RFC o la CURP, para ligarlo con el trámite. **(RN02)**
6	El sistema valida que el RFC, si es una persona moral, se encuentre dado de alta en el padrón de establecimientos, para el caso de una persona física ingresa el RFC o CURP, si no lo encuentra dentro del padrón de particulares lo enviará hacia una pantalla de captura para complementar los datos del usuario. **(EX02) (EA01)**
7	Ventanilla acepta la relación del establecimiento o el particular con el trámite.
8	El sistema genera el número de trámite.
9	Ventanilla solicita la impresión del volante que sirve como acuse del usuario para comprobar que ingresó su trámite y con este mismo posteriormente regresará por la respuesta
10	El sistema imprime el volante, el cual una parte se entrega al usuario y la otra se anexa junto con la documentación entregada.
11	Fin del caso de uso.

Escenarios Alternos.

EA01 –El usuario que está ingresando el trámite es una persona física y no está dentro del padrón de establecimientos del sistema.	
Paso	**Acción**
1	El sistema detecta que se desea ingresar un trámite y asociarlo a una persona física, que no se encuentra previamente dentro del padrón de particulares..
2	El sistema presenta la pantalla de captura para los datos del particular y así registrarlo dentro del padrón de establecimientos como particular.
3	Ventanilla captura los datos del particular.
4	El sistema continúa con la asociación del nuevo particular registrado al trámite.
5	Fin del caso de uso

Excepciones

EX01 –El trámite no se encuentra dentro del catálogo de trámites que atiende la Comisión.	
Paso	**Acción**
1	El sistema determina qué trámite no se puede ingresar debido a que no existe dentro del catálogo de trámites para ser atendidos por la Comisión.
2	El sistema le informa al usuario que no se puede ingresar el trámite y regresa a la pantalla inicial de ingreso de trámites.
3	Fin del Caso de Uso.

EX02 –El establecimiento (persona moral), no se encuentra dado de alta dentro del padrón de establecimientos de la Comisión.	
Paso	**Acción**
1	El sistema determina qué trámite no se puede ingresar debido a que no está dado de alta el establecimiento que lo solicita.
2	El sistema le informa a Ventanilla que no encuentra al establecimiento y regresa a la pantalla inicial para ingresar el trámite.
3	Fin del Caso de Uso.

Reglas de Negocio

Id	**Regla de Negocio**
RN01	Para que un trámite pueda ser ingresado al sistema, éste debe estar previamente dado de alta dentro del catálogo de trámites que atiende la Comisión, dentro de este catálogo un trámite se compone de los siguientes elementos: • Formato • Tipo de Trámite • Subtipo de Trámite • Modalidad
RN02	Para que una persona moral pueda ingresar cualquier trámite previamente debe de estar dado de alta dentro del padrón de establecimientos. En el caso de ser una persona física no es necesario tenerlo dentro del padrón, sin embargo, en el momento de la solicitud del trámite se capturan sus datos.

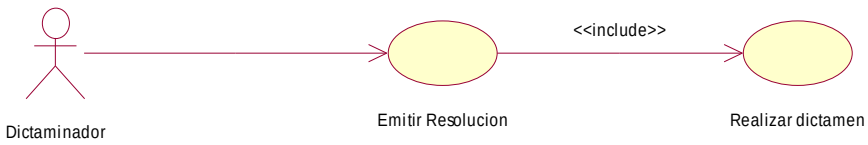
Requerimientos No-Funcionales

Id	**Requerimientos No-Funcionales**
1	NA

5.3.3 Emitir Resolución.

Generales del Caso de Uso		
Nombre Caso Uso	Emitir Resolución	
Objetivo		
Emitir la resolución de un trámite en particular, la resolución es la respuesta de la Comisión hacia la solicitud hecha por un usuario respecto a un tipo de trámite, ésta puede ser positiva, negativa o prevención generalmente.		
Nivel del Caso de Uso	**Prioridad**	**Complejidad**
Nivel 1 (Caso de Uso de Usuario)	Alta	Alta
Actores involucrados		
Dictaminador		
Evento Disparador (Trigger)		
El Dictaminador requiere emitir la resolución de un trámite que le fue asignado.		
Precondiciones		
El trámite debe estar recibido por el área, para que se pueda emitir su resolución. De esto se encarga el usuario de Mesa de Control.		
Post-condiciones		
Se libera la resolución, se imprime y el trámite será devuelto al Centro Integral de Servicios. **EX01, EX02,** El trámite permanece sin resolución.		

Diagrama de Caso de Uso



Escenario Principal.

Paso	Acción
1	El Dictaminador ingresa al panel de Resoluciones.
2	El sistema le presenta el panel de resoluciones.
3	El usuario ingresa el número de trámite para emitir una resolución.
4	El sistema valida que el trámite se encuentre en un estatus válido para emitir la resolución. **(EX01)**
5	El usuario ingresa el dictamen para justificar la resolución del trámite**.(RN01)**
6	El sistema valida que el dictamen sea válido. **(EX02)**
7	El usuario genera la resolución y la imprime.
8	Fin del caso de uso.

Excepciones

EX01 –El trámite no se encuentra en un estado válido para la resolución.	
Paso	**Acción**
1	El sistema determina que el estado del trámite no es válido para emitir la resolución.
2	El sistema le informa al usuario que el trámite no es válido y lo regresa al punto inicial para capturar un nuevo número.
3	Fin del Caso de Uso.

EX02 – El dictamen no es válido.	
Paso	**Acción**
1	El sistema determina que el dictamen aplicado no es válido para el trámite.
2	El sistema informa al usuario que el sistema no es válido y lo regresa a capturar de nuevo un dictamen o en su defecto a modificarlo
3	Fin del Caso de Uso.

Reglas de Negocio

Id	Regla de Negocio
RN01	Para que un trámite pueda tener una resolución, se debe de relacionar con un dictamen de acuerdo al tipo de éste, y debe cumplir con lo siguiente: • El dictamen deberá ser válido, es decir, debe de contener los elementos necesarios para poder emitir una resolución.

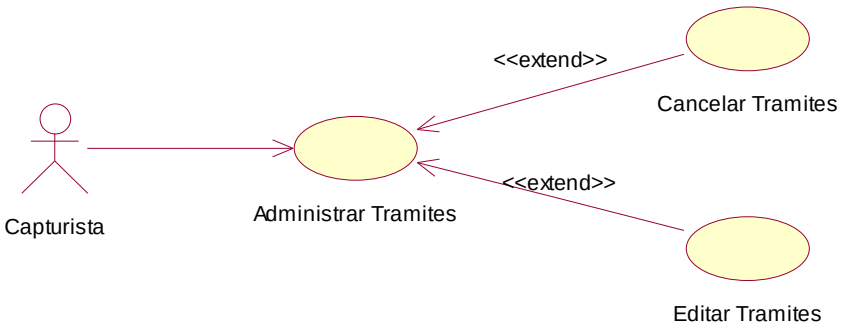
Requerimientos No-Funcionales

Id	Requerimientos No-Funcionales
1	NA

5.3.4 Administrar Trámites.

Generales del Caso de Uso		
Nombre Caso Uso	Administrar Trámites.	
Objetivo		
Administrar los trámites que ingresan a la Comisión, es decir, capturar su información complementaria, cancelar en caso de ser necesario, modificar captura. Cabe mencionar que también se pueden modificar establecimientos y particulares, ya que el ingreso de éstos en el sistema también se ve como un trámite.		
Nivel del Caso de Uso	**Prioridad**	**Complejidad**
Nivel 1 (Caso de Uso de Usuario)	Alta	Alta
Actores involucrados		
Capturista		
Evento Disparador (Trigger)		
El capturista requiere editar información sobre un trámite o algún establecimiento a través de un trámite.		
Precondiciones		
El trámite debe haber sido ingresado previamente dentro del sistema y a través de esto se debió de haber generado un número de trámite que es con el cual se identifica lo que queremos modificar.		
Post-condiciones		
Una vez terminado este proceso y de acuerdo a lo que se haya realizado, se turna al área técnica si es necesaria una resolución, si no lo requiere se cierra el caso. **EX01,** El trámite permanece sin modificaciones.		

Diagrama de Caso de Uso



Escenario Principal.

Paso	Acción
1	El capturista ingresa al panel de Administración de trámites.
2	El sistema le presenta el panel de Administración de trámites.
3	El capturista ingresa el número de trámite para editarlo o cancelarlo.
4	El sistema valida que el trámite se encuentre en un estatus válido para modificarlo. **(EX01) (RN01)**
5	El sistema presenta al usuario las opciones que puede aplicar sobre el trámite**.**
6	El capturista selecciona la opción deseada(editar, cancelar)
7	El sistema valida que la opción seleccionada sea aplicable al trámite.
8	Si la opción editar es válida el sistema presenta la pantalla para editar el trámite. **(EA01)**
9	El capturista edita la información requerida e indica que terminó con la edición.
10	El sistema guarda las modificaciones realizadas al trámite.
11	El sistema informa al capturista que se guardaron las modificaciones.
12	Fin del caso de uso

Escenarios Alternos.

EA01 –El Capturista selecciona la opción Cancelar Trámite.	
Paso	**Acción**
1	El sistema detecta que el capturista seleccionó la opción para cancelar un trámite.
2	El sistema valida que efectivamente el trámite se encuentre en un estatus válido para poder cancelarlo. **(RN02)**
3	El capturista cancela el trámite dando los motivos de cancelación.
4	El sistema pregunta si efectivamente se desea cancelar el trámite.
5	El usuario confirma la cancelación.
6	El sistema confirma la cancelación al capturista.
7	Fin del caso de uso

Excepciones

EX01 –El trámite no se encuentra en un estado válido para editarlo.	
Paso	**Acción**
1	El sistema determina que el estado del trámite no es válido para editarlo.
2	El sistema le informa al usuario que el trámite no es válido y lo regresa al punto inicial para capturar un nuevo número.
3	Fin del Caso de Uso.

Reglas de Negocio

Id	**Regla de Negocio**
RN01	Para editar un trámite debe de encontrarse en un estatus válido, en el caso de que el trámite ya tenga asociada una resolución no se puede editar, igualmente si el trámite ya fue entregado al usuario tampoco se puede editar.
RN02	Un trámite se puede cancelar siempre y cuando no tenga una resolución asociada y no haya sido entregado al usuario.

Requerimientos No-Funcionales

Id	**Requerimientos No-Funcionales**
1	NA

5.3.5 Controlar Trámites.

Generales del Caso de Uso		
Nombre Caso Uso	Controlar Trámites.	
Objetivo		
Controlar el flujo de los trámites entre el Centro Integral de Servicios (CIS) y las áreas técnicas para finalmente entregar al usuario la resolución a su solicitud.		
Nivel del Caso de Uso	**Prioridad**	**Complejidad**
Nivel 1 (Caso de Uso de Usuario)	Alta	Alta
Actores involucrados		
Mesa de Control		
Evento Disparador (Trigger)		
El usuario de Mesa de Control requiere recibir o entregar trámites.		
Precondiciones		
El trámite o trámites que va a recibir o entregar Mesa de Control, se debe de encontrar en un estado válido para cada una de estas opciones.		
Post-condiciones		
Una vez que el trámite es entregado o recibido cambia de estatus, de acuerdo al flujo previamente establecido. **EX01,** No se modifica ningún trámite.		

Diagrama de Caso de Uso

Escenario Principal.

Paso	Acción
1	Mesa de Control ingresa a la opción de entregar trámites. **(EA01)**
2	El sistema le presenta el panel de entrega de trámites, en donde se mostraran el total de trámites por área que tiene pendientes por entregar. **(RN01)**
3	Mesa de Control, selecciona el área a la que desea entregar trámites. **(EX01)**

4	El sistema desplegara el detalle de todos los trámites que la Mesa de Control tiene pendientes por entregar, con respecto al área seleccionada.
5	Mesa de Control seleccionara los trámites que pretende entregar**.**
6	El sistema emitirá el resumen de los trámites seleccionados y le pedirá al usuario la confirmación para alterar estos trámites e imprimir el acuse.
7	Mesa de Control acepta el resumen.
8	El sistema altera los trámites seleccionados.
9	El sistema imprime el acuse para que éste sirva como control documental sobre las entregas de trámites entre áreas.
10	Fin del caso de uso.

Escenarios Alternos.

EA01 – Mesa de Control selecciona la opción recibir trámites.	
Paso	**Acción**
1	El sistema le presenta el panel de recepción de trámites, en donde se mostraran el total de trámites por área que tiene pendientes por recibir.
2	Mesa de Control selecciona el área a la que desea recibir trámites. **(EX01)**
3	El sistema desplegará el detalle de todos los trámites que la Mesa de Control tiene pendientes por entregar, con respecto al área seleccionada.
4	Mesa de Control seleccionará los trámites que pretende entregar**.**
5	El sistema emitirá el resumen de los trámites seleccionados y le pedirá al usuario la confirmación para alterar estos trámites e imprimir el acuse.
6	Mesa de Control acepta el resumen.
7	El sistema altera los trámites seleccionados.
8	El sistema altera los trámites seleccionados.
9	El sistema imprime el acuse para que éste sirva como control documental sobre la recepción de trámites entre áreas.
10	Fin del caso de uso.

Excepciones

EX01 –El trámite no se encuentra para ser entregado o recibido.	
Paso	**Acción**
1	El sistema, al presentar la pantalla de recepción o la de entrega no muestra el trámite deseado.
2	Si el usuario tiene otros trámites para entregar o recibir los selecciona en la pantalla, si no es así simplemente sale de la opción.
3	Fin del Caso de Uso.

Reglas de Negocio

Id	**Regla de Negocio**
RN01	La mayoría de los trámites que atiende la Comisión, deben pasar por el siguiente flujo: Ingreso-Captura-Entrega del CIS al área-Recepción del área-Entrega del área al CIS-Recepción del CIS-Entrega al Usuario Aunque algunos tipos de trámites especiales como son Avisos simplemente se quedan en el Ingreso y allí finaliza su ciclo.

Requerimientos No-Funcionales

Id	**Requerimientos No-Funcionales**
1	NA

5.3.6 Liberar Resolución.

Generales del Caso de Uso		
Nombre Caso Uso	Liberar Resolución.	
Objetivo		
El Mando Medio que puede ser un gerente, subdirector o director de área a través de este mecanismo autoriza la resolución que ha sido preparada por el dictaminador.		
Nivel del Caso de Uso	**Prioridad**	**Complejidad**
Nivel 1 (Caso de Uso de Usuario)	Alta	Media
Actores involucrados		
Mando Medio		
Evento Disparador (Trigger)		
El Mando Medio una vez que revisó la resolución desea liberarla, es decir, autorizarla para que sea ésta la respuesta hacia el usuario.		

Precondiciones
La resolución para generarla previamente debió de haber tenido un dictamen.
Post-condiciones
La resolución se libera. El trámite ya no puede ser modificado de ninguna forma. El verificador ya no puede alterar la resolución. **EX01,** La resolución aún no ha sido liberada.

Diagrama de Caso de Uso

Escenario Principal.

Paso	Acción
1	Mando Medio ingresa al panel para liberar resoluciones.
2	El sistema le presenta el panel para liberar resoluciones.
3	Mando Medio ingresa el número de trámite para ver su resolución. **(EX01)**
4	El sistema despliega la resolución asociada al trámite que previamente fue generada por el Dictaminador, así como el detalle del dictamen
5	Mando Medio revisa el proceso y libera la resolución. **(EA01)**
6	El sistema solicita la confirmación para liberar la resolución.
7	Mando Medio acepta.
8	El sistema libera la resolución. **(RN01)**
10	Fin del caso de uso.

Escenarios Alternos.

EA01 – Mando Medio no libera la resolución.	
Paso	**Acción**
1	El sistema no libera la resolución, por lo tanto el trámite se regresa al Dictaminador, para que éste la modifique hasta que obtenga el visto bueno, y asimismo la liberación por parte del Mando Medio
2	Fin del caso de uso.

Excepciones

EX01 –El trámite aun no tiene una resolución asociada.	
Paso	**Acción**
1	El sistema no encuentra una resolución asociada al número de trámite.
2	Si sistema informa al usuario que no se encontró una resolución asociada al trámite.
3	Fin del Caso de Uso.

Reglas de Negocio

Id	**Regla de Negocio**
RN01	Para que un trámite pueda ser entregado del área al CIS, requiere tener una resolución asociada y que ésta haya sido liberada, ya que es la respuesta que se le entregará al usuario final.

Requerimientos No-Funcionales

Id	**Requerimientos No-Funcionales**
1	NA

5.3.7 Entregar Resolución.

Nombre Caso de uso	Entregar Resolución.
Actores	Ventanilla
Tipo	Primario, Esencial
Descripción	El usuario Ventanilla, entrega la resolución de un trámite al usuario final, con esto el trámite llega al fin de su ciclo.

Diagrama de Caso de Uso

5.3.8 Consultar Trámites.

Nombre Caso de uso	Consultar Trámites.
Actores	Mesa de Control
Tipo	Primario, Esencial
Descripción	El usuario Mesa de Control consulta el estatus de un trámite, es decir en que parte se encuentra dentro del flujo.

Diagrama de Caso de Uso

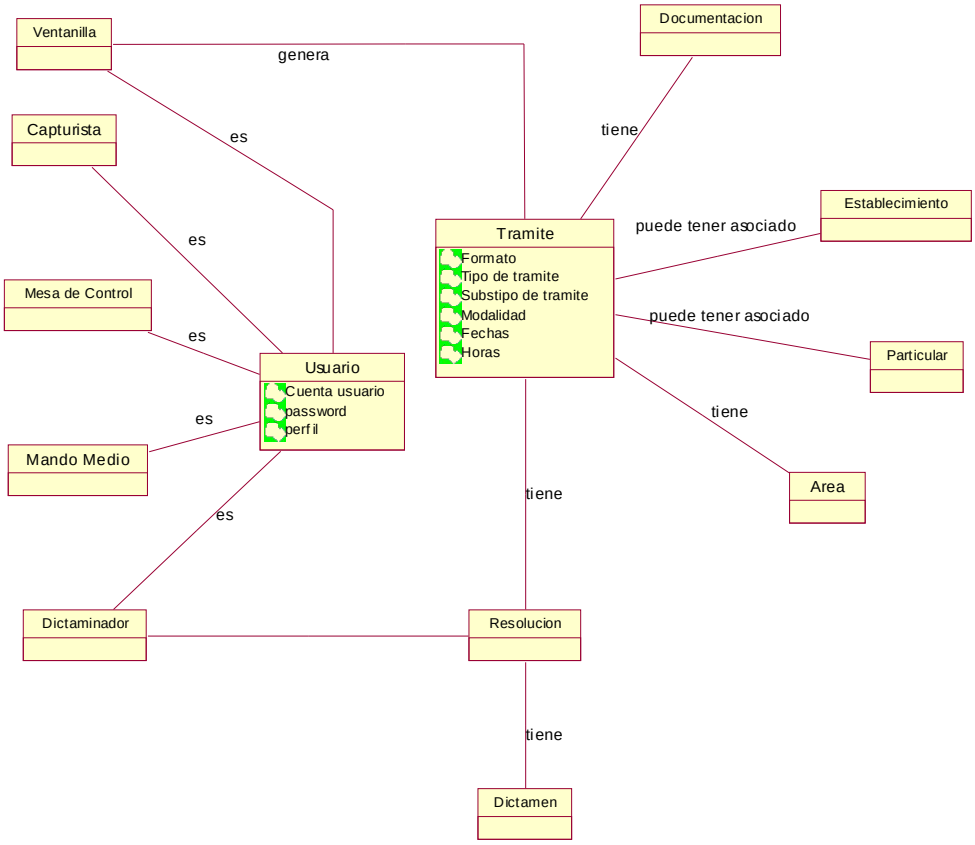
5.3.9 Consultar Reportes.

Nombre Caso de uso	Consultar Reportes.
Actores	Mando Medio.
Tipo	Primario, Esencial
Descripción	El usuario Mando Medio consulta los diferentes reportes que ofrece el sistema, que pueden ser estadísticos, gerenciales, de seguimiento o de operación diaria.

Diagrama de Caso de Uso

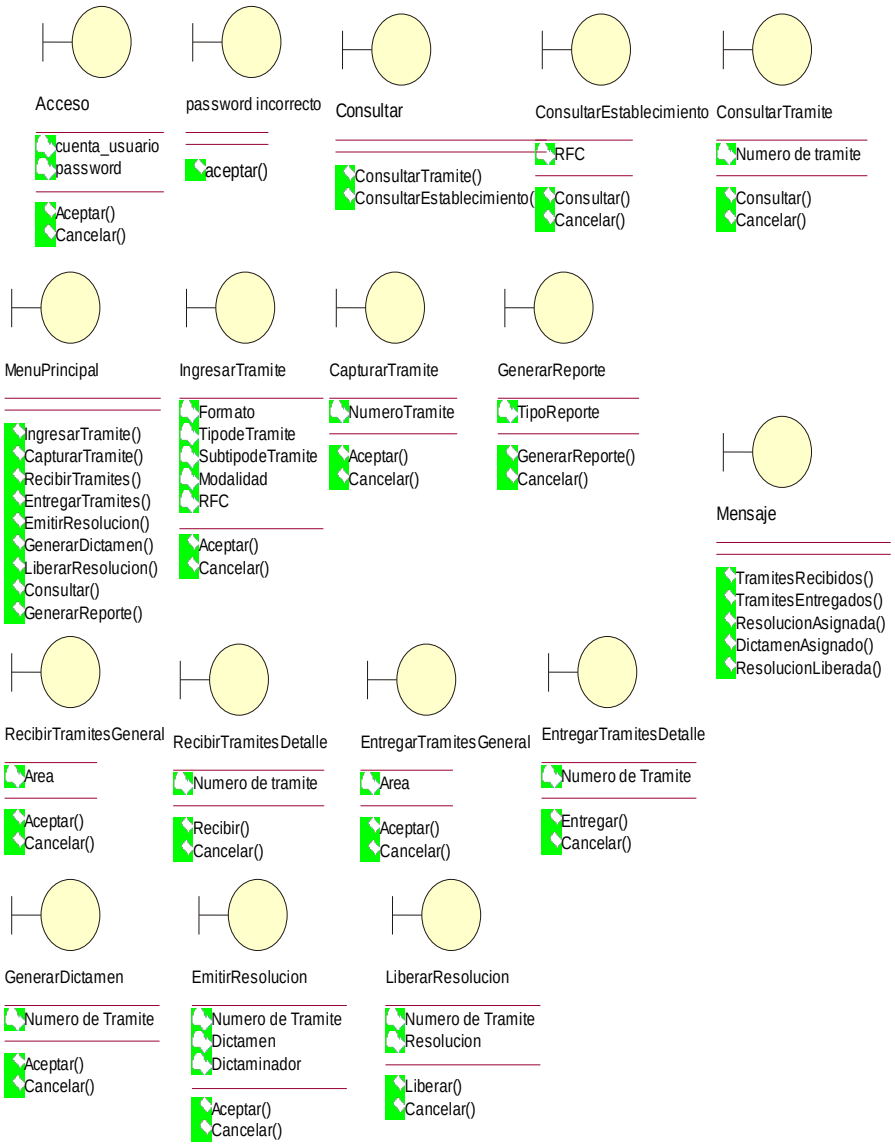
5.4 MODELO CONCEPTUAL.

Actores	Clases	Atributos
Ventanilla	Trámite	Cuenta usuario
Capturista	Establecimiento	Password
Mesa de Control	Particular	Perfil
Dictaminador	Dictamen	Formato
Mando Medio	Documentación	Tipo de trámite
Usuario	Resolución	Subtipo de trámite
	Padrón	Modalidad
	Número de trámite	Forma de entrada
	Área	Fechas
		Horas

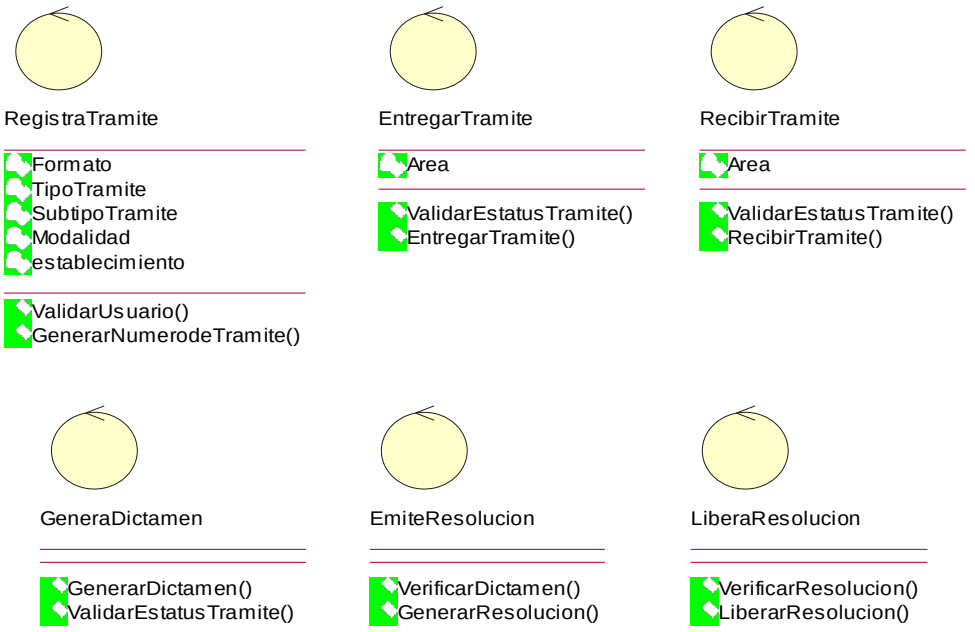


5.5 DIAGRAMA DE CLASES ANÁLISIS

5.5.1 Boundary

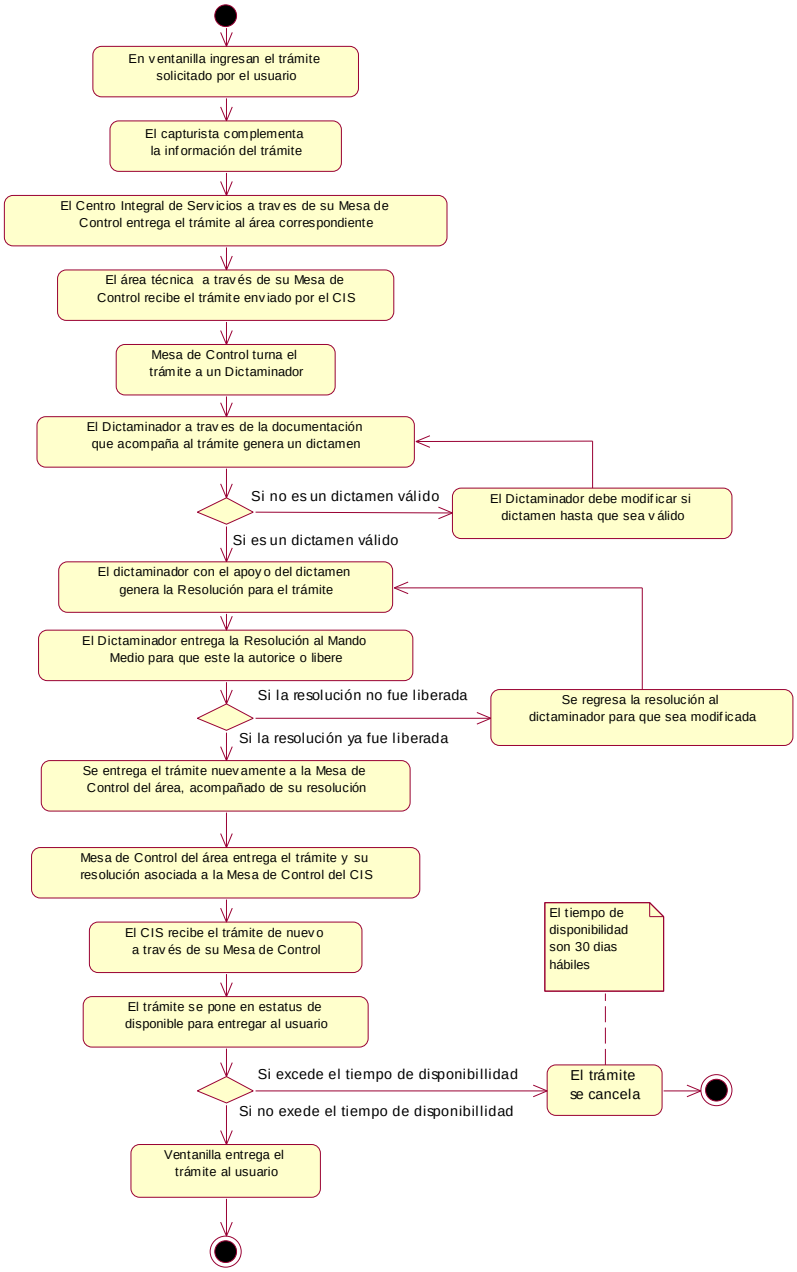


5.5.2 Control



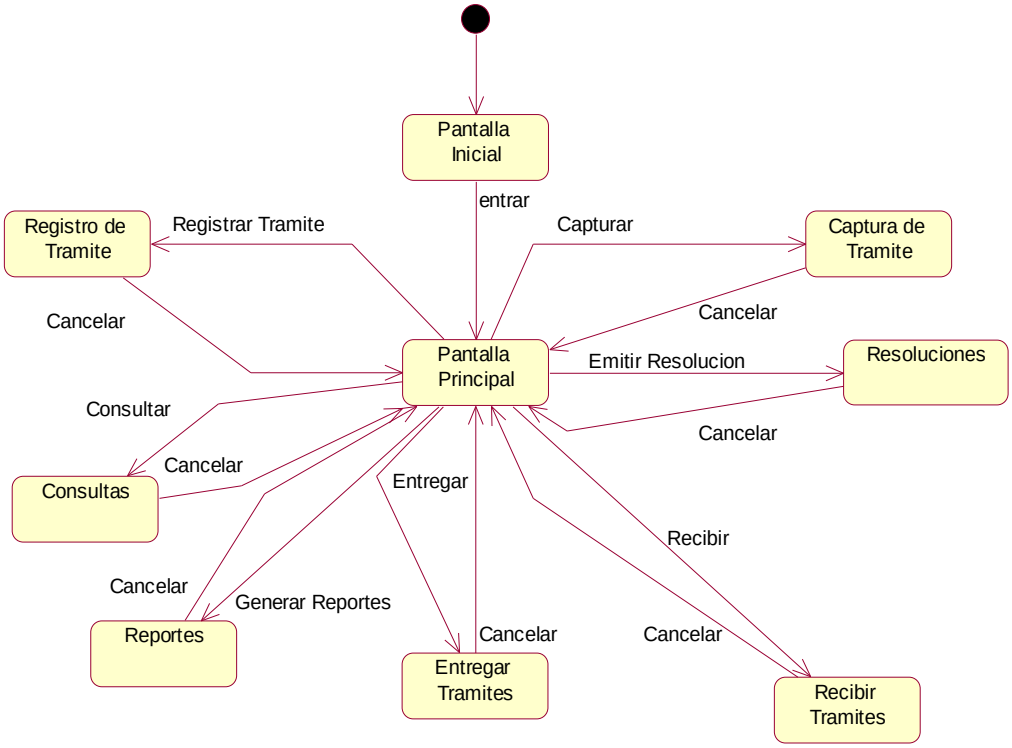
5.5.3 Entidad

5.6 DIAGRAMA DE ACTIVIDADES.

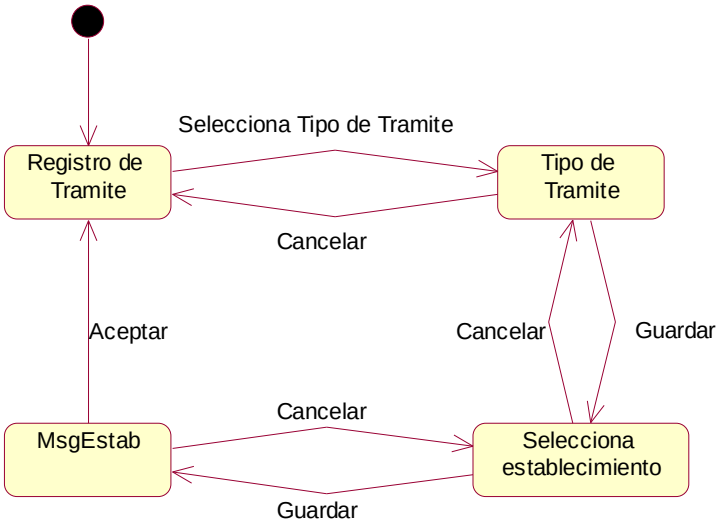


5.7 DIAGRAMAS DE ESTADOS (NAVEGACIÓN).

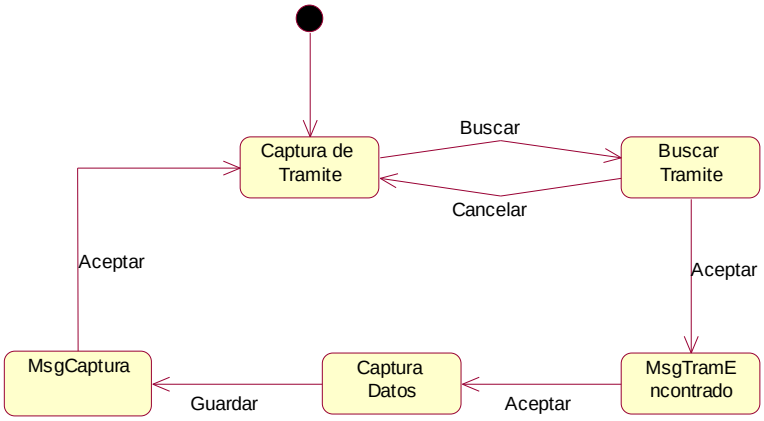
5.7.1 Diagrama de Navegación Principal.



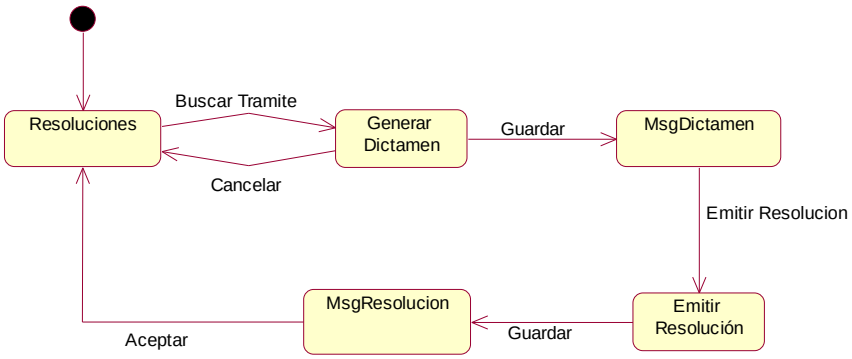
5.7.2 Registro de Trámite.



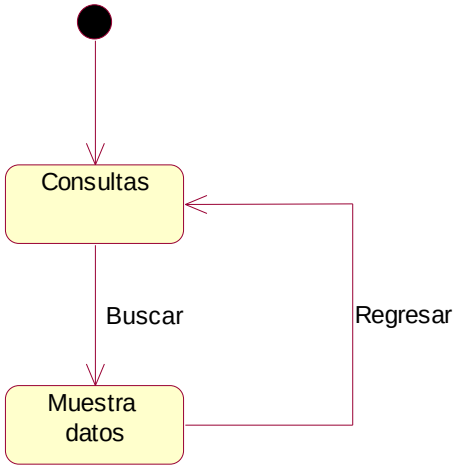
5.7.3 Captura de Trámite.



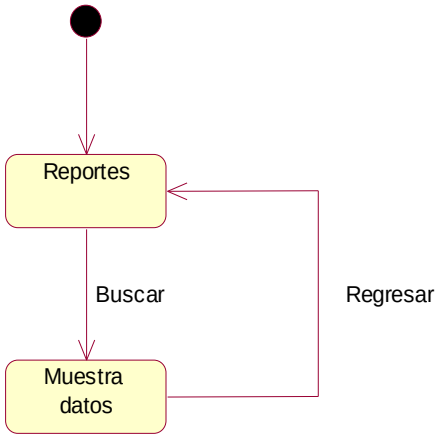
5.7.4 Resoluciones.



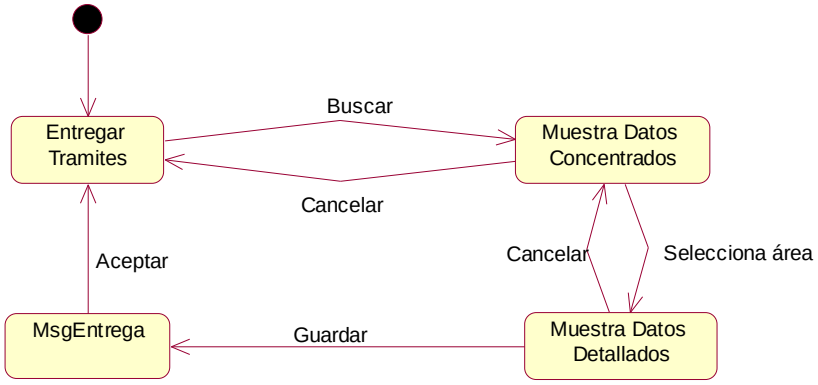
5.7.5 Consultas.



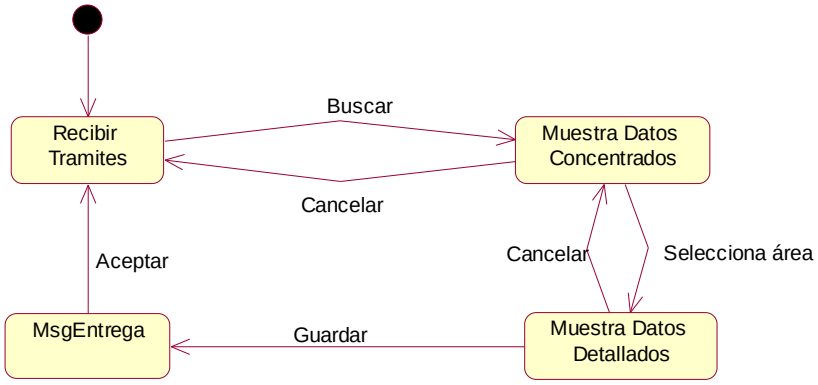
5.7.6 Reportes.



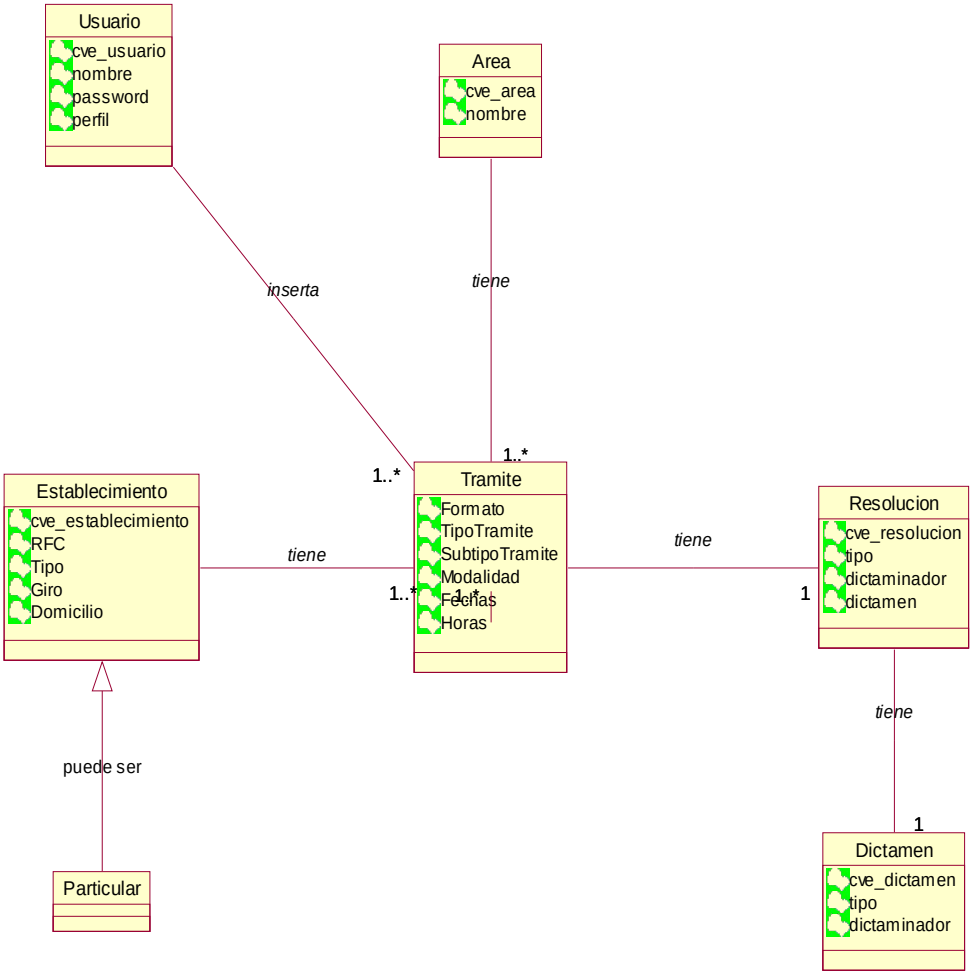
5.7.7 Entregar Trámites.



5.7.8 Recibir Trámites

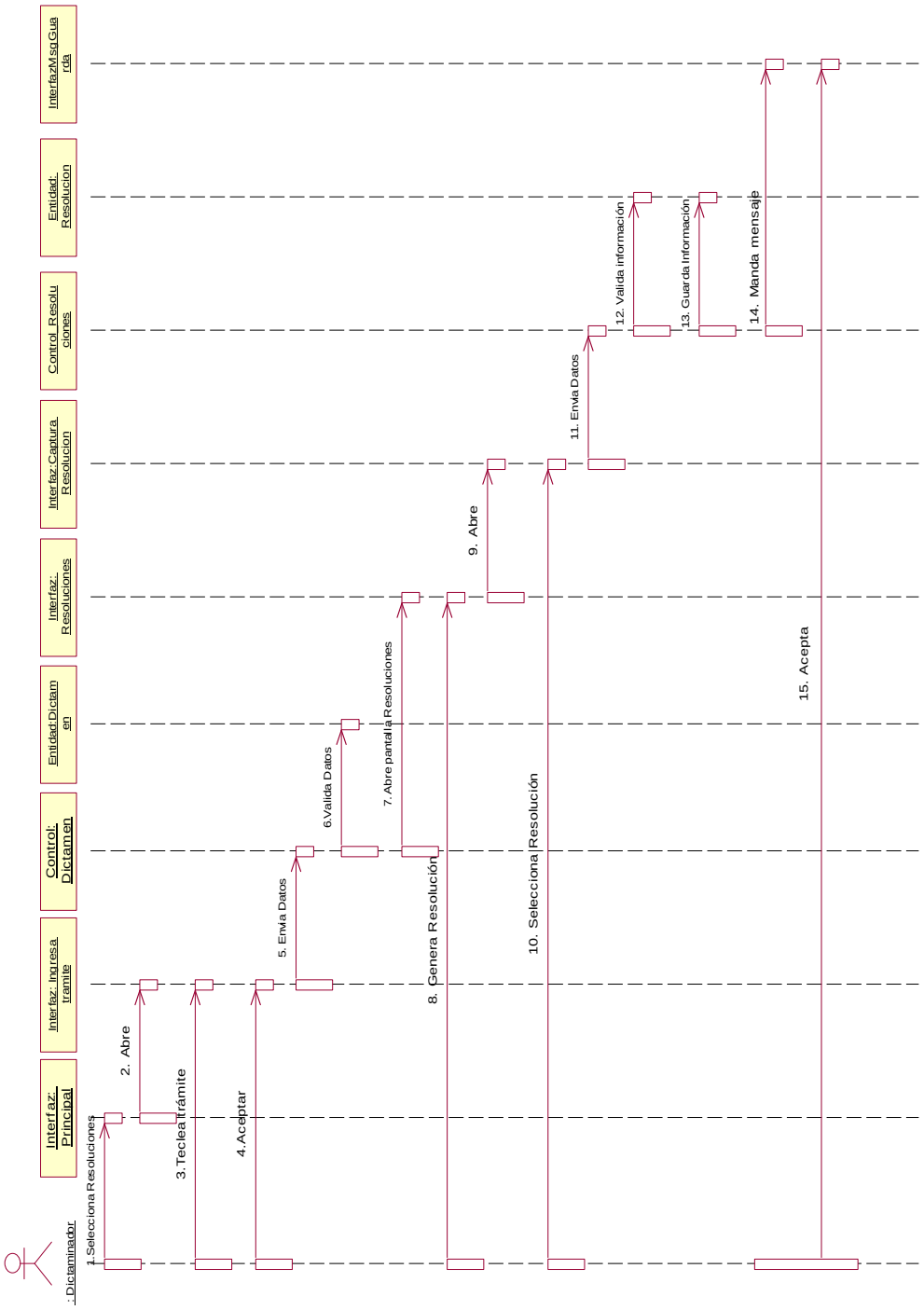


5.8 DIAGRAMA DE CLASES DISEÑO.

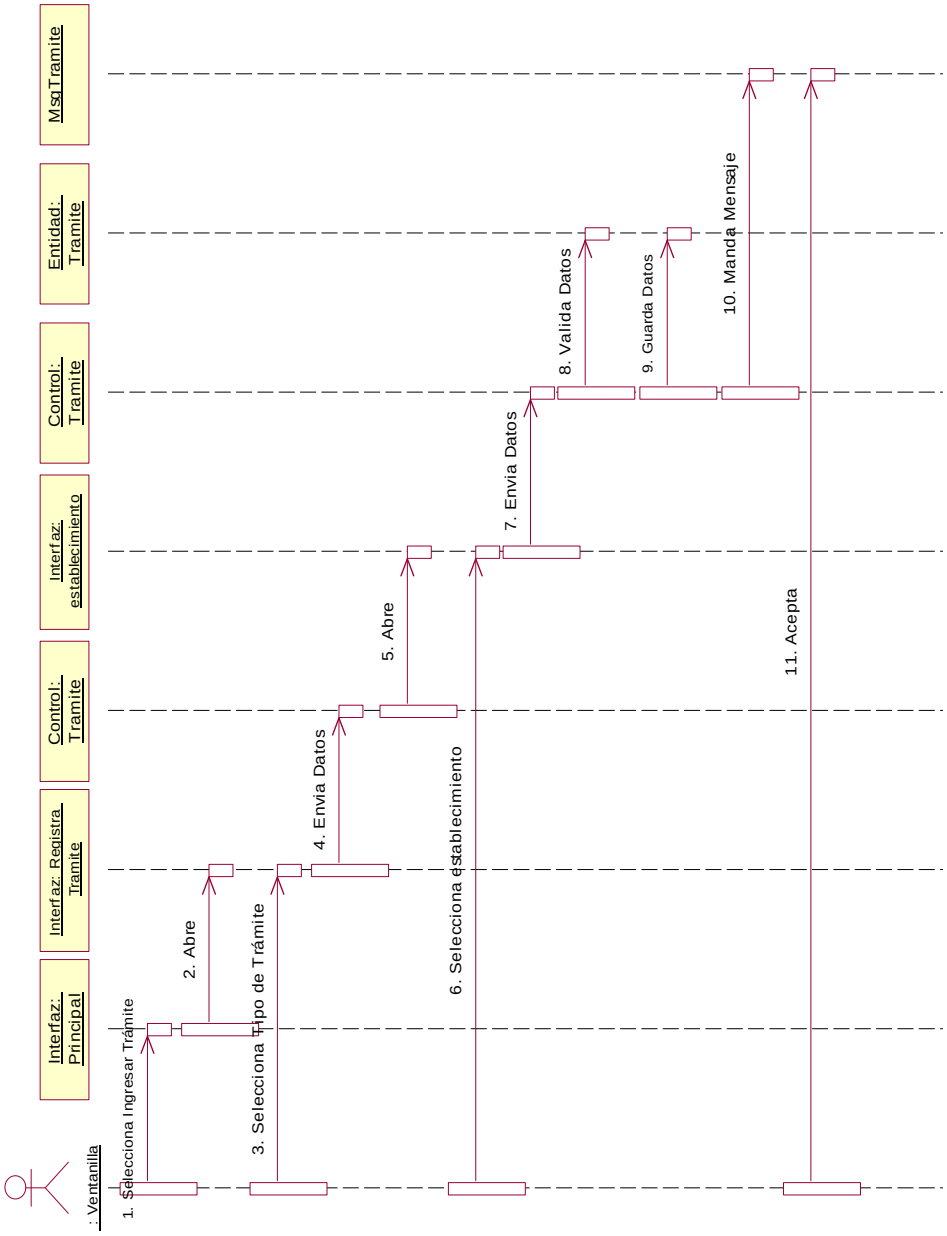


5.9 DIAGRAMAS DE SECUENCIA.

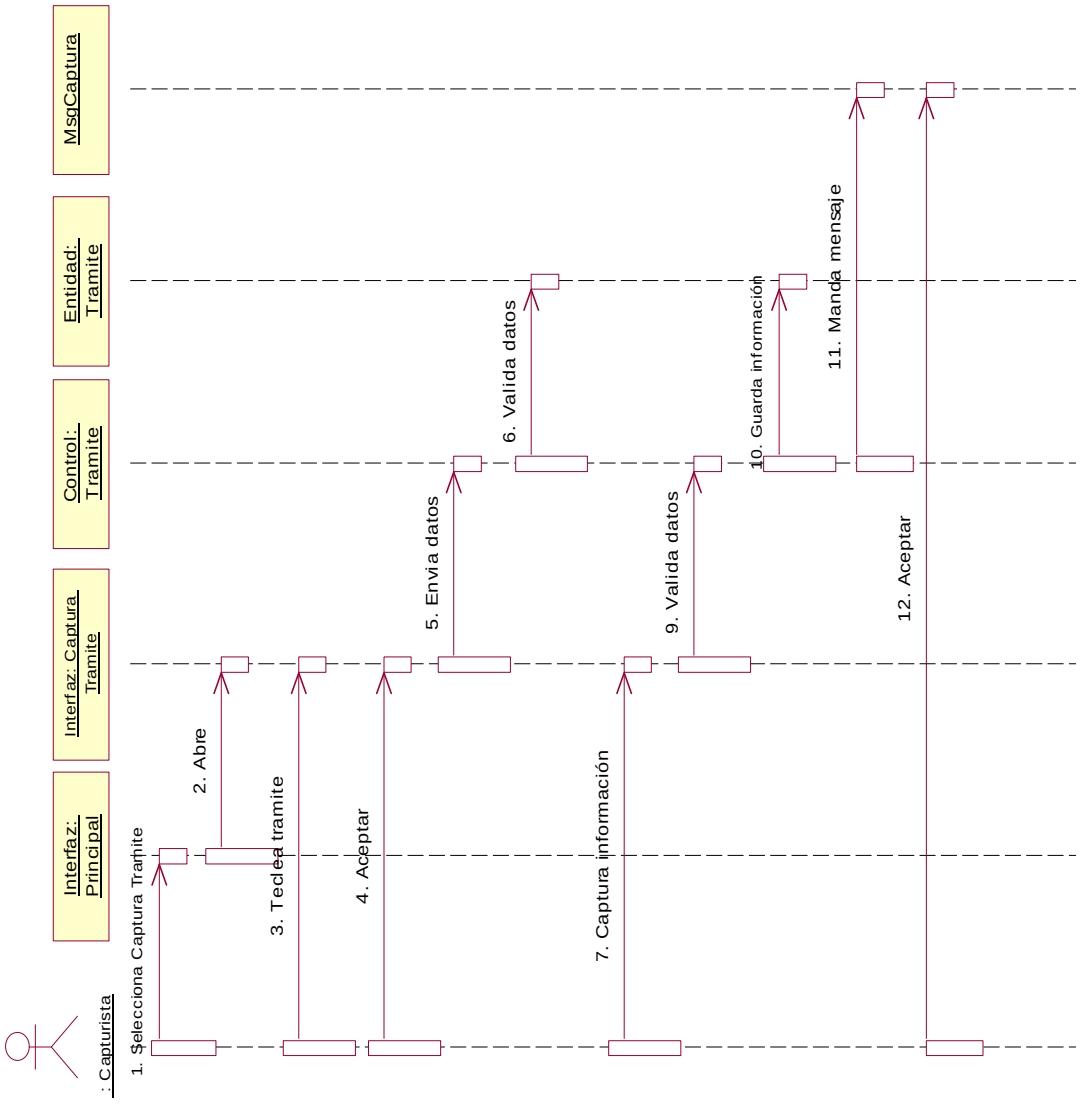
5.9.1 Resoluciones.



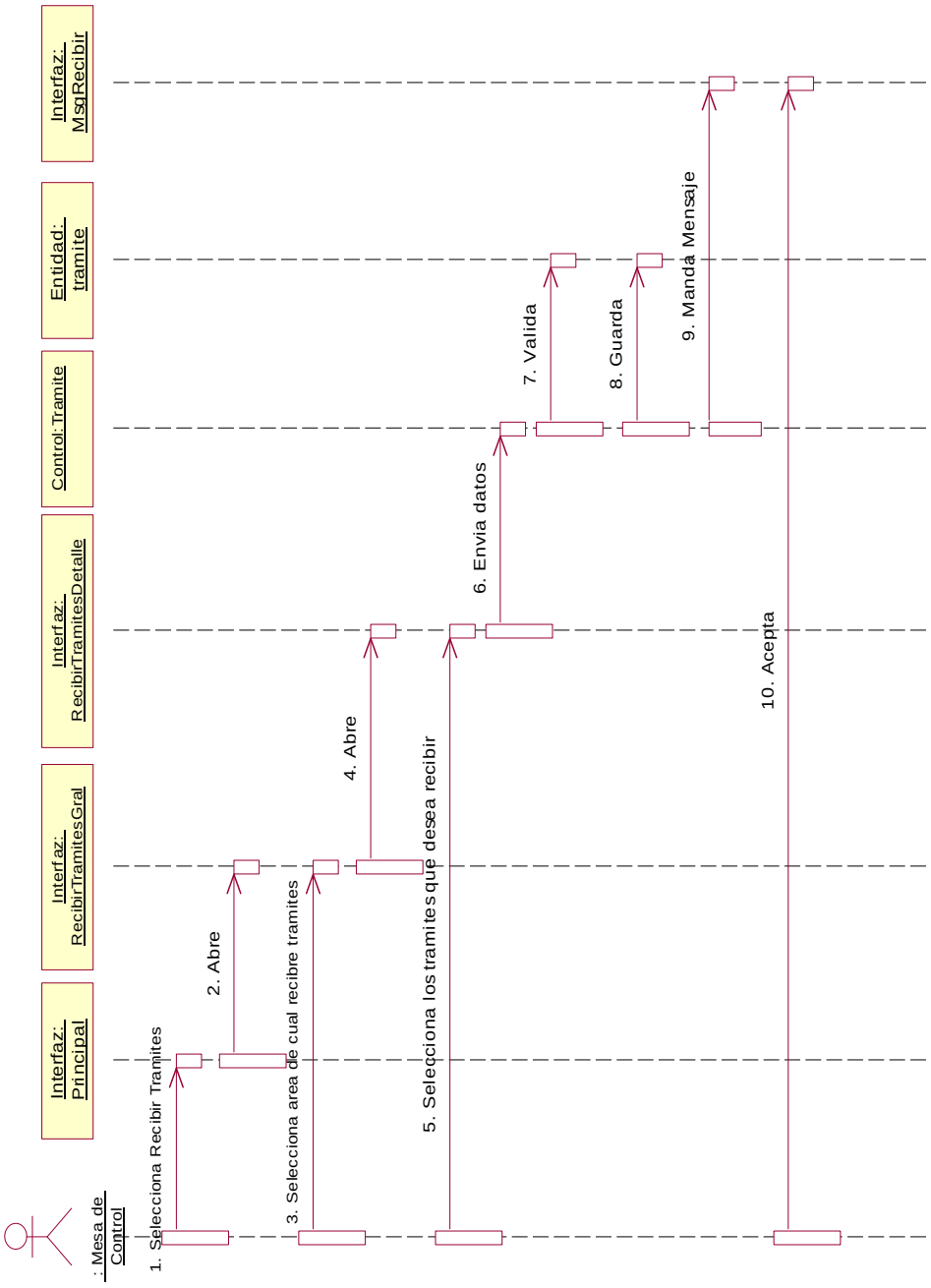
5.9.2 Ingresar Trámites.



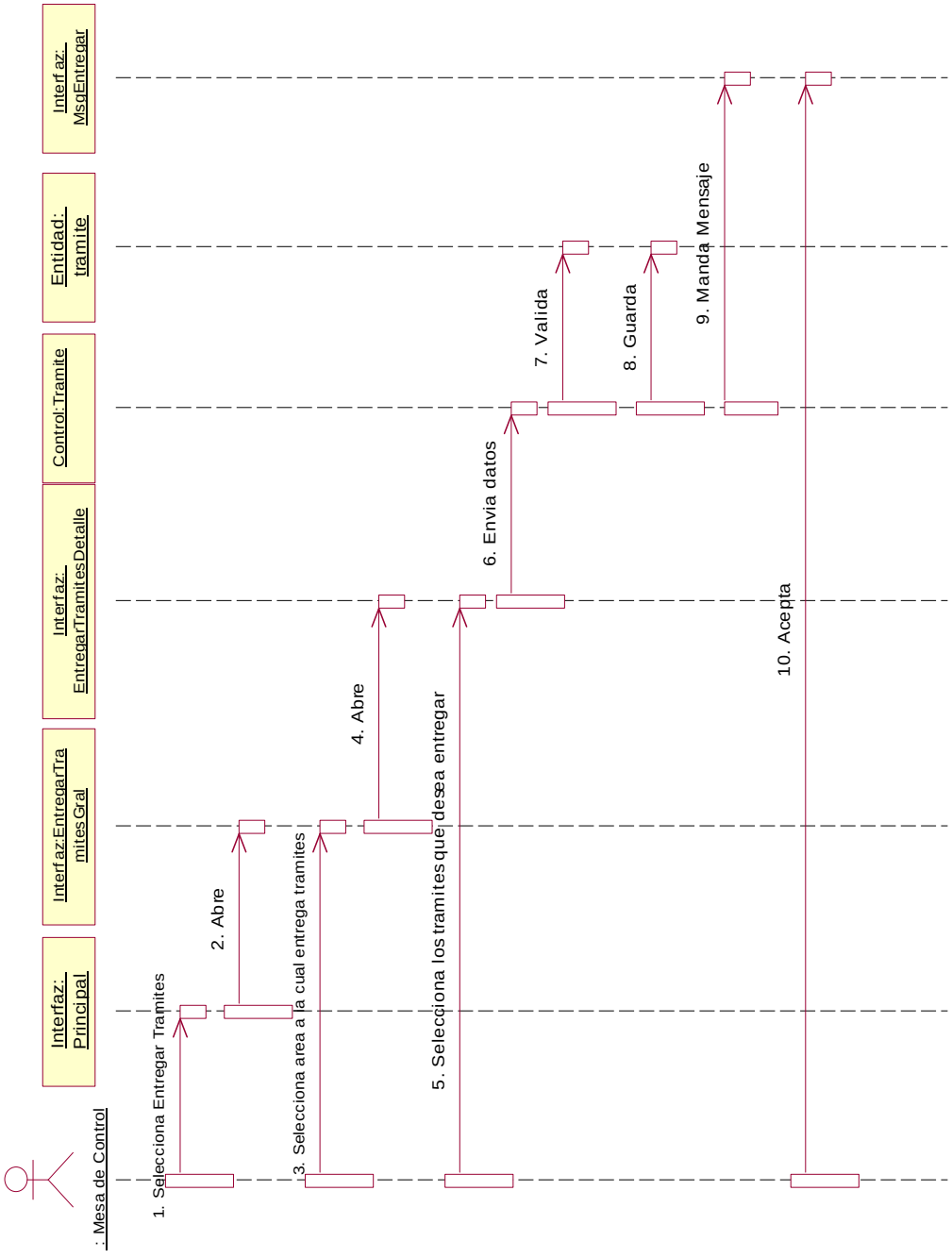
5.9.3 Capturar Trámites.



5.9.4 Recibir Trámites.



5.9.5 Entregar Trámites.



5.10 LISTADOS DE CÓDIGO FUENTE.

A continuación se ejemplifica el diagrama de secuencia para recibir trámites, en la parte donde se despliegan los trámites de manera general, con los principales archivos que intervienen con el framework struts.

En el archivo struts-config.xml, tenemos el mapeo de las acciones y sus propiedades, para este caso como se muestra en el siguiente listado:

```
<action-mappings>

    <action
    path="/recibirTramitesGral"
            type="com.sitracofepris.struts.controller.RecibirTramitesGralAction"
            name="RecibirTramitesForm"
            scope="request">
        <forward name="success" path="/pages/RecibirTramitesGral.jsp"/>
        <forward name="failure" path="/error.jsp"/>
    </action>

</action-mappings>
<controller processorClass="org.apache.struts.tiles.TilesRequestProcessor"/>
<message-resources parameter="com/ApplicationResource"/>
```

El siguiente listado muestra la clase Action que es el controlador para este caso:

```
package com.sitracofepris.struts.controller;

import ...

/**...*/
public class RecibirTramitesGralAction extends org.apache.struts.action.Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {

        HttpSession session = request.getSession(true);

        try {
            RecibirTramitesForm recibirTramitesForm = (RecibirTramitesForm) form;

            ServiciosRecibirTramites servicios = new ServiciosRecibirTramites();
            List listaTramites = null;

            listaTramites = servicios.obtenListaRecibirTramitesGral();
            recibirTramitesForm.setListaTramites(listaTramites);
        } catch (Exception e) {
            e.printStackTrace();
            return mapping.findForward("failure");
        } catch (Throwable e) {
            e.printStackTrace();
        }

        // Devolvemos el Action Forward para el caso Success
        return mapping.findForward("success");
    }
```

Dentro de la clase Action se llama a la clase de tipo ActionForm que se muestra a continuación:

```
package com.sitracofepris.struts.view;

import java.util.List;

/**
 *
 * @author DAVID
 */
public class RecibirTramitesForm extends org.apache.struts.action.ActionForm {
  private List listaTramites;

    public List getListaTramites() {
        return listaTramites;
    }

    public void setListaTramites(List listaTramites) {
        this.listaTramites = listaTramites;
    }

}
```

También dentro de la clase Action se invoca a los servicios que representan la capa del modelo dentro del framework struts que es la que se encarga de interactuar con la Base de Datos, a continuación se muestra la parte principal de esta clase:

```
        try {
            // Componemos la sentencia SQL para obtener los Aspirantes.
            String query = "select b.nom_largo as area,count(*) as tramites " +
                    " from hist_tram a inner join area b on a.cve_area=b.cve_area " +
                    " where cve_estado_tramite=2 and a.ingreso > '20080101' group by b.nom_largo";
            // Ejecutamos la query y obtenemos el resultado.
            Statement stmt = conn.createStatement();
            resultado = stmt.executeQuery(query);
            String area;
            int tramites;
            while (resultado.next()) {

                area = resultado.getString("area");
                tramites = resultado.getInt("tramites");

                RecibirTramitesGralVO tramitesGralVO = new RecibirTramitesGralVO();
                tramitesGralVO.setArea(area);
                tramitesGralVO.setTramites(tramites);

                listaTramitesGral.add(tramitesGralVO);
            }
            return listaTramitesGral;
        } catch (SQLException se) {
            System.err.println("Se ha producido un error de BD.");
            System.out.println("ERROR AL OBTENER LA LISTA DE TRAMITES POR RECIBIR GENERAL");
            se.printStackTrace();
            return null;
        } finally {
            conn.close();
        }
```

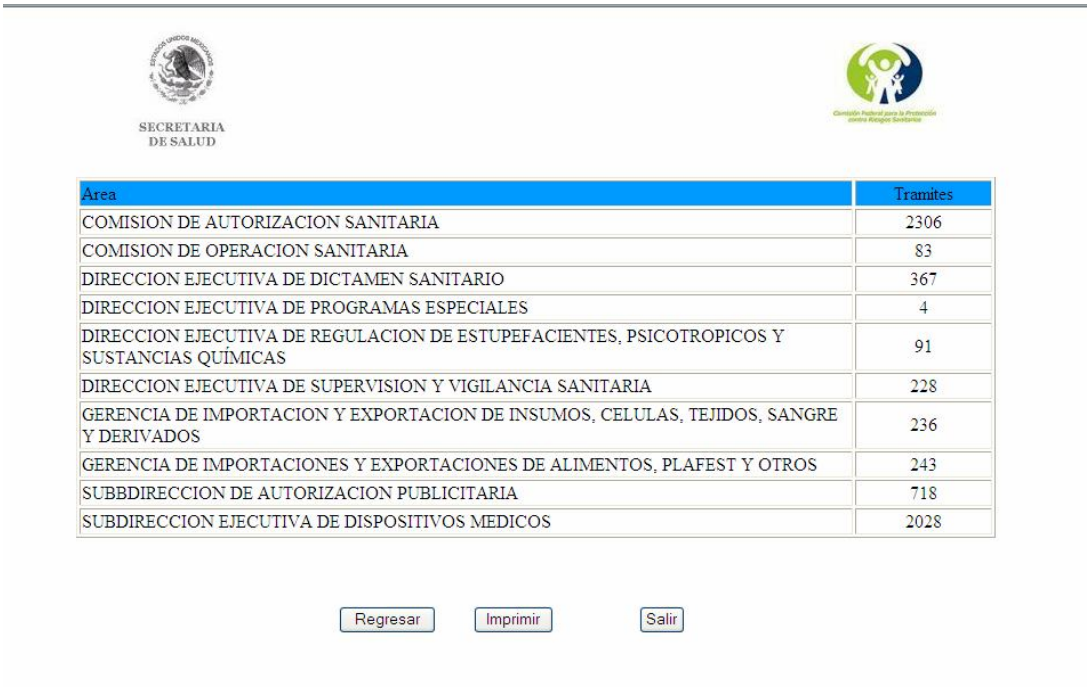
En los servicios mandamos llamar a la clase de tipo RecibirTramitesGralVO, esta clase contiene los métodos accesores (getters y setters) para los datos, como se muestra a continuación:

```
package com.sitracofepris.vo;

import java.io.Serializable;

/**
 *
 * @author DAVID
 */
public class RecibirTramitesGralVO implements Serializable{
    private String area;
    private int tramites;

    public String getArea() {
        return area;
    }

    public void setArea(String area) {
        this.area = area;
    }

    public int getTramites() {
        return tramites;
    }

    public void setTramites(int tramites) {
        this.tramites = tramites;
    }

}
```

Finalmente se muestra la capa de la vista dentro del framework struts que es el JSP que se muestra al usuario, en el siguiente listado se muestra la parte principal del código de esta pagina:

```
<head>
        <title>Accesos</title>
<script language="Javascript" src="./js/funciones.js"></script>
</head>

<body bgcolor="#FFFFFF">

<jsp:include page="/pages/Encabezado.jsp"/>

<br>

<table align="center" width="50%" border="1">
  <tr>
        <td width="40%" bgcolor="#0099FF">
      <div align="center"></div>
      <bean:message key="mensaje.accesos.Fecha"/></td>
        <td width="30%" bgcolor="#0099FF">
      <div align="center"></div>
      <bean:message key="mensaje.accesos.Hora"/></td>
  </tr>
  <logic:notEmpty name="RecibirTramitesForm" property="listaTramites">
        <logic:iterate id="tramites" name="RecibirTramitesForm" property="listaTramites" type="com.sitracofepris.vo.RecibirTramitesGralVO">
        <tr>
                <td><bean:write name="tramites" property="area"/></td>
                <td><bean:write name="tramites" property="tramites"/></td>
        </tr>
        </logic:iterate>
 </logic:notEmpty>
</table>
```

Una vez que tenemos estos este conjunto de archivos sincronizados dentro de la aplicación, a continuación se muestra la salida ya en un navegador para esta funcionalidad, cabe mencionar que esta pagina es un previo del sistema ya que están pendientes por definir los estilos y logotipos que se utilizaran para el desarrollo.



CONCLUSIONES.

Las ventajas que tiene utilizar un framework de trabajo como lo es struts para desarrollar este sistema tiene muchas ventajas, el hecho de separar el desarrollo en diferentes capas nos permite dar un mantenimiento mas fácil al sistema; los trámites que atiende la comisión pueden variar en cuanto a la información solicitada constantemente por motivos de simplificación, también puede cambiar el formato de las resoluciones de trámites por motivos de seguridad, sin embargo estos cambios resultan relativamente sencillos ya que por estar separada por niveles la aplicación solamente afectamos en este caso a la capa de la vista sin tener que afectar toda la aplicación.

Al principio al utilizar un framework se puede tornar complicado el entender como funciona y contemplar todos los elementos que lo componen, sin embargo una vez que superamos esto, el desarrollo de la aplicación se vuelve mucho mas fácil y rápido ya que tenemos bien identificado que pasos debemos de seguir para programar cualquier modulo, además la reutilización de código se vuelve mayor.

Se puede concluir que el sistema para la atención de trámites para la Comisión Federal para la Protección contra Riesgos Sanitarios, cubre las necesidades del usuario, además de que facilitara la integración con otros sistemas y le dará un valor agregado a la institución al facilitar y agilizar la atención de trámites que atiende.

El sistema cumple con el principal objetivo de la institución que es el de centralizar la información que le compete a COFEPRIS, para que se puedan tomar decisiones oportunas sobre cualquier contingencia.

Al final de este trabajo considero que queda de manifiesto la formación que me dio la Universidad, en donde gracias a esta he podido llegar a la resolución de problemas tanto de instituciones publicas como privadas.

BIBLIOGRAFÍA Y FUENTES CONSULTADAS.

Deitel, H.M y Deitel, P.J
Como Programar en Java
Prentince Hall
México 1988

Richard Hightower
Jakarta Struts Live
Source Beat
E.U.A. 2004

Joseph Schuller
Aprendiendo UML en 24 horas
Pearson Educación
México 2000

Kendall & Kendall
Analisis y Diseño de Sistemas
Pearson Educación
México 1998

Ley General de Salud
www.salud.gob.mx

Como utilizar el Modelo Vista Controlador
http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html