UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 3

Título: Extensión del sistema GINA para satisfacer los requisitos de control de acceso en el Registro Central de la Aduana General de la República de Cuba.

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas.

Autor: Leodan Rodriguez Diaz

Tutor: Ing. Raymond Weeden Gamboa

Tutor: Ing. Yasel Antonio Romero Piñeiro

La Habana, junio de 2012

"Año 54 de la Revolución"

Declaración de Autoría

Declaro ser autor del presente trabajo y reconozco a la Universidad de las Ciencias Informáticas dueña de los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____

Leodan Rodriguez Diaz

Firma del Autor

_____

Ing. Raymond Weeden Gamboa

Firma del Tutor

_____

Ing. Yasel Antonio Romero Piñeiro

Firma del Tutor

Agradecimientos

A todos los que de alguna forma hicieron posible la realización de este trabajo.

A mis padres, mi familia y mi novia por su apoyo incondicional, y por estar presente en cada momento en que se ha hecho necesario.

A mis tutores por su entrega, apoyo, consejos y paciencia.

A todos mis compañeros de proyecto y brigada por todos estos años de amistad y ayuda mutua.

A todos ¡Muchas Gracias!

Dedicatoria

A mis padres por traerme a este mundo, por hacer de mí un hombre de bien, por su ejemplo, por enseñarme a luchar por lo que quiero.

A mi madre, por ser la mejor madre del mundo, por su ejemplo y espíritu positivo y de lucha, por dedicarme cada momento, cada acto y cada pensamiento.

A mi familia por ser un ejemplo de ello, por ser fieles centinelas de mi presente y futuro, por apoyarme en cada momento.

A mi novia, por todos estos años juntos, por todo su amor y dedicación, por hacer de mí el hombre más feliz del mundo, por ser lo mejor que me ha pasado en la vida y claro, por darme los mejores suegros que alguien desearía tener.

A mis amigos de ahora y de toda la vida, porque las experiencias con ellos compartidas son también parte de mi vida.

A mis compañeros de grupo y proyecto, por todos estos años de mutuo aprendizaje y apoyo.

¡A todos por ser parte de este momento tan importante en mi vida!

Resumen

El correcto manejo y la seguridad de la información, es un factor fundamental para toda institución, más aún cuando su funcionamiento depende en alta medida de esta, pues una incorrecta manipulación, desvío o pérdida de la misma ocasionaría daños económicos y sociales irreversibles. Para impedir la ocurrencia de estos sucesos, es necesario implementar medidas de administración, seguridad y control de la información. La Aduana General de la República posee un sistema informático para el control de sus procesos, toda la información que se maneja en este es de suma importancia, por lo que su seguridad es una premisa esencial en su funcionamiento. A su vez, esta información resulta en varios casos necesaria para diferentes procesos relacionados con entidades externas a los procesos aduaneros, es precisamente este uno de los elementos que hoy no se garantiza por dicho sistema, a pesar de ser un requerimiento indispensable.

El sistema actualmente en explotación posee un subsistema de administración capaz de gestionar la seguridad de los usuarios pertenecientes a la Aduana General de la República, pero no de gestionar los usuarios pertenecientes a entidades que se relacionan con la misma. Teniendo en cuenta este elemento, y dado el necesario intercambio de información de la Aduana General de la República con dichas entidades y sus trabajadores, se hace necesario extender los subsistemas de Administración y el de Registro Central, logrando en su conjunto los requerimientos especificados. Lo abordado anteriormente, forma la propuesta que se hace en el presente trabajo, en el cual se hará además un estudio de varios elementos de seguridad de sistemas, de técnicas y herramientas en el desarrollo de software, con el objetivo final no de crear nuevos sistemas, sino de crear nuevas funcionalidades a los actuales en pos de lograr lo requerido por la Aduana General de la República.

Tabla de Contenidos

Introducción

Garantizar la seguridad y protección de la sociedad socialista y de la economía nacional, así como la recaudación fiscal y la emisión de las estadísticas del comercio exterior, a través del cumplimiento de las políticas estatales de competencia aduanera para el tráfico internacional de viajeros, mercancías y medios de transporte, son las tareas principales de la Aduana General de la República (AGR). (1)

La ARG fue fundada el 5 de febrero de 1963, constituye el órgano que vela y regula el control aduanero aplicable a la entrada, el tránsito, el cabotaje, el trasbordo, el depósito y la salida del territorio nacional de mercancías, viajeros y sus equipajes, bienes y valores sujetos a regulaciones especiales, así como los medios en los que se transporten. Constituye además parte de la Administración Central del Estado y se subordina al Consejo de Ministros. (1)

Dado el auge actual de las Tecnologías de la Información y las Comunicaciones (TIC), las entidades que manejan gran cantidad de información y procesos, se han sumado a una carrera de informatización de sus mecanismos de trabajo, la AGR es una de ellas, pues con el crecimiento continuo actual del comercio internacional y la interdependencia económica que vincula y norma las relaciones entre estados, se hace necesario para una buena aplicación de las políticas aduaneras, contar con un personal altamente calificado junto a una tecnología y sistemas adecuados, que se correspondan con las necesidades actuales.

Con el surgimiento de la Universidad de las Ciencias Informáticas (UCI) en el año 2002, se abrieron nuevas puertas a varias instituciones nacionales y extranjeras para su incursión en la informatización de sus procesos, entre ellas se encuentra la AGR, la cual a pesar de tener el apoyo del Centro de Automatización para la Dirección y la Información (CADI)[1], necesitaba de un impulso mayor en la producción de un sistema para gestionar sus procesos, sistema con el que se pretende mantener a la AGR como referencia de mejora, perfeccionamiento, automatización, seguridad y confianza para todas las aduanas del mundo.

Para la realización de diversos procesos aduaneros, como el comercio exterior e interior, se requiere la participación de varias entidades autorizadas con la AGR, las cuales deben ser gestionadas y controladas. Para realizar con seguridad y estricto control estas actividades, la AGR mantiene un registro de las

[1] CADI: Centro de Automatización para la Dirección y la Información, centro dedicado al desarrollo de soluciones informáticas para la aduana, en él se concentran los especialistas de la rama y que tienen precisamente como propósito la creación de aplicaciones que intervengan en los diferentes procesos aduaneros.

personas naturales o físicas (personas de existencia visible, real, física o natural que intervienen en los distintos procesos); o jurídicas o morales (empresas, entidades, con las cuales se relaciona la AGR), que han sido autorizadas por los organismos competentes a realizar actividades de comercio exterior, de las que en la ejecución de las actividades que tienen autorizadas se vinculan con la autoridad aduanera, así como de aquellas que son autorizadas por la AGR a realizar determinadas actividades relacionadas con la importación y la exportación de mercancías durante el proceso de despacho, con la finalidad de obtener sus datos y características fundamentales, así como asignar los códigos necesarios para acceder a los sistemas automatizados de despacho (2).

Para ello, en la AGR opera el Registro Central de Entidades (RCE), encargado de llevar a cabo los procesos que se derivan de las normas establecidas para realizar estas actividades. El RCE actualmente hace uso de 2 herramientas para su funcionamiento, las cuales son detalladas a continuación:

Para la generación de reportes, emplean DataEase 4.0, que data de 1988 y actualmente implica incompatibilidades de software con los Sistema Operativos, aplicaciones y herramientas modernas. Esta utiliza un lenguaje procedural, el cual carece de las capacidades que poseen los lenguajes actuales que usan estándares Structure Query Language (SQL); este sistema requiere de previos conocimientos informáticos y de programación para la realización de varias acciones, lo que es poco amigable al usuario estándar, el cual al interactuar con la aplicación puede omitir detalles importantes para el negocio, además de provocar el crecimiento exponencial del trabajo a realizar cuando los procedimientos se hacen complejos.

El otro sistema usado es el Sistema Único de Aduanas (SUA), el cual está en uso desde el año 2000. Dicho sistema automatiza los cinco procesos aduaneros (despacho comercial, no comercial, de viajeros, de medios de transporte y de postal y envío), pero el módulo destinado a la gestión de personas y entidades del RCE no cumple a cabalidad sus objetivos. En el SUA son insertados los datos generados por el DataEase y este es capaz, dado que cuenta con tablas de control, de generar otros tipos de reportes necesarios en los procesos aduaneros; pero esto crea dependencia y redundancia en el proceso.

Dichas soluciones no cuentan con una administración de trabajadores o entidades, así como con un sistema de autenticación y control de acceso que garantice la seguridad de las operaciones con ellos o por ellos realizados, elemento que se hace de vital importancia en las aplicaciones informáticas actuales y que en el presente caso incide directamente en que no se garantice el acceso a la información del RCE desde fuera de instituciones aduaneras cubanas o extranjeras, tanto dentro como fuera de Cuba; y se

hace aún más evidente y necesario, pues para la realización de diferentes tareas en el flujo de procesos aduaneros, se requiere la participación de entidades y trabajadores externos a la AGR que están registrados en el RCE, por lo que se hace necesario gestionar los mismos de manera eficaz y segura, así como la autorización a ejecutar dichas tareas en dependencia de su rol en el proceso; además de permitir a través de mecanismos la participación en los procesos y la consulta de datos necesarios desde fuera de instituciones aduaneras e incluso de la nación.

Como fruto de la cooperación entre la UCI y la AGR, durante el año 2010 en el Centro para la Informatización y la Gestión de Entidades (CEIGE), se desarrolló un sistema para la gestión de las entidades en el RCE y otro para la administración y gestión de usuarios y roles dentro del sistema, pero dados nuevos requerimientos y especificaciones capturadas en nuevos análisis al negocio aduanero, así como el surgimiento de nuevas propuestas de desarrollo para la interacción de las entidades externas con la AGR, además de no garantizar la seguridad, la autenticación y el control de acceso al sistema desarrollado por parte de los sistemas de administración implementados en el CEIGE, se determina que los sistemas ya desarrollados no solucionan las nuevas especificaciones del sistema en desarrollo.

De manera general se identificaron como principales dificultades la incompatibilidad de los sistemas usados en el RCE con los Sistemas Operativos y aplicaciones modernas y el uso de las máquinas virtuales como vía de solución a este problema, lo que trae consigo la lentitud y las trabas en el trabajo del usuario. Existe además redundancia de información, debido a que los datos de trabajadores y entidades deben repetirse al cambiar de estado. La interfaz de usuario es poco intuitiva y amigable, dado que el DataEase se usa en interfaz de línea de comandos[2].

Se identificaron además tareas en el trabajo aduanal que no son resueltas o automatizadas con los sistemas actuales, como la consulta de datos necesarios desde el exterior por parte de trabajadores de entidades. Otra de las deficiencias detectadas y que requiere vital atención es la ausencia de un mecanismo de autenticación y control de acceso de usuarios, evidenciándose fallas de seguridad y confiabilidad en el sistema. Por último se identifica una alta incompatibilidad en tecnología y arquitectura de los sistemas en uso con el Sistema de Gestión Integral Aduanera (GINA)[3].

[2] Interfaz de Línea de Comandos. CLI, Interfaz de modo texto para la ejecución de tareas mediante comandos en texto simple que el Sistema Operativo y las aplicaciones en uso son capaz de interpretar.

[3] Gestión Integral de Aduanas. GINA, sistema desarrollado por el Centro de Soluciones para la Aduana, del Centro para la Informatización y la Gestión de Entidades de la Universidad de las Ciencias Informáticas, como colaboración de la UCI y la Aduana General de la República.

Una vez analizada la problemática anterior, se determina el siguiente **problema a resolver**: el sistema del RCE de la AGR no garantiza la gestión del acceso de los trabajadores externos.

Para dar solución al problema planteado anteriormente, se define como **objeto de estudio:** el mecanismo de autenticación y control de acceso del sistema GINA.

Se determina como **campo de acción:** el mecanismo de autenticación y control de acceso en el sistema del RCE de la AGR.

Se define además como **objetivo general**: la creación de un mecanismo que garantice la gestión del acceso de los trabajadores externos a la AGR y como **objetivos específicos**:

- Establecer el marco conceptual de referencia.
- Determinar los requerimientos del sistema.
- Modelar la solución propuesta.
- Implementar la solución.
- Validar la solución.

Para alcanzar el objetivo trazado, se definen como principales **tareas de la investigación:**

- Recopilación de la bibliografía referente al tema.
- Selección de la bibliografía.
- Análisis de la bibliografía.
- Estudio de las principales soluciones que manejen el problema en cuestión.
- Desarrollo del modelo de especificación de requisitos.
- Definición de la solución teniendo en cuenta la arquitectura del sistema.
- Descripción formal de la solución de acuerdo a los procedimientos del departamento.
- Definición del modelo de diseño de la solución.
- Desarrollo del modelo de implementación.
- Definición de las pruebas de validación.
- Validación de la solución desarrollada.
- Presentación de la solución.

Concluido este trabajo y cumpliéndose las tareas determinadas y con ellos los objetivos trazados, **se espera como resultado** un mecanismo que gestione y controle el acceso de los trabajadores externos a la AGR.

Durante el desarrollo de este trabajo son utilizados algunos métodos científicos de la investigación que ayudan a guiar exitosamente el desarrollo de la solución.

- Histórico-lógico**:** con el objetivo de realizar un estudio del estado del arte del tema a investigar. De esta manera se puede conocer acerca de la existencia y características de sistemas de este tipo que hayan sido desarrollados anteriormente.
- Analítico-sintético**:** con el objetivo de analizar teorías y documentos, y a partir de ello realizar un estudio previo acerca de las características y la necesidad de un componente que facilite la búsqueda del historial de personas naturales en la AGR. (3)
- Modelación**:** con el objetivo de realizar abstracciones para dar una explicación de la realidad existente. Su condición principal es la relación entre el modelo y el objeto que se modela, que en este caso sería el mecanismo en cuestión.
- Implementación**:** con el objetivo fundamental de desarrollar el mecanismo modelado, implementando las clases que fueron modeladas durante la etapa de diseño. (4)

El presente trabajo está estructurado en tres capítulos, los cuales son descritos a continuación:

- Capítulo 1**:** abarca lo relacionado con la Fundamentación Teórica de la investigación. Se enuncian los principales conceptos que se abordan en la investigación. Se realiza además un estudio de las soluciones al problema planteado ya existentes, así como de las herramientas, metodologías y lenguajes a usar en el desarrollo de la solución.
- Capítulo 2**:** en este capítulo se aborda el Análisis y el Diseño de la solución, describiendo a su vez varios artefactos que se generan durante dichas fases. Se describe además la definición de los requerimientos del sistema, el diseño de los prototipos de interfaz de usuario, los diagramas de secuencia, de paquetes, de clases del negocio, entre otras puntualizaciones.
- Capítulo 3**:** implementación y validación de la solución propuesta; teniendo como base los artefactos concebidos en las etapas de análisis y diseño para la implementación de la solución y posteriormente haciendo uso de las distintas técnicas de evaluación de software, para determinar posibles no conformidades así como saber si se cumplió o no con los objetivos propuestos.

Capítulo 1. Fundamentación Teórica

1.1. Introducción.

La información se ha convertido en uno de los más grandes tesoros para el hombre en estos tiempos, más aún cuando de esta depende el funcionamiento correcto de flujos de trabajo y sistemas. El almacenamiento y seguridad de la información necesaria es una de las principales premisas de una entidad moderna, más aún cuando esta se encuentra y se genera en grandes volúmenes. En la actualidad una de las ideas más defendidas cuando de grandes volúmenes de información y su seguridad se trata, es la de un Registro Central (RC), en el cual se archiva la información necesaria para su funcionamiento e interacción con otras entidades, información que por ende necesita de acertadas y estrictas medidas de seguridad y protección, alcanzadas únicamente mediante efectivos sistemas de seguridad.

En el presente capítulo se analizan varios sistemas que gestionan Registros Centrales (RC) así como la seguridad de su información mediante sistemas de autenticación y control de acceso, haciendo énfasis en sistemas de entidades aduaneras. Se hace una descripción de los principales conceptos que fundamentan el contenido de la investigación y se detallan además elementos importantes acerca de las tecnologías y sistemas empleados para el desarrollo del sistema propuesto.

1.2. Registro Central de Entidades de la Aduana General de la República.

Según los vocablos y la sintaxis que lo constituyen y su significado definido por la Real Academia de la Lengua Española, se puede definir un RC como un archivo o registro de datos centralizado, que se usa con el objetivo de establecer una base de datos completa y unificada (5). El RCE de la AGR es un registro de las personas naturales o físicas (personas de existencia visible, real, física o natural que intervienen en los distintos procesos); o jurídicas o morales (empresas, entidades, con las cuales se relaciona la AGR), que han sido autorizadas por los organismos competentes a realizar actividades de comercio exterior, de las que en la ejecución de las actividades que tienen autorizadas, se vinculan con la autoridad aduanera así como de aquellas que son autorizadas por la AGR a realizar determinadas actividades relacionadas con la importación y la exportación de mercancías durante el proceso de despacho, con la finalidad de obtener sus datos y características fundamentales, así como asignar los códigos necesarios para acceder a los sistemas automatizados de despacho (2).

1.3. Seguridad de sistemas.

Para mantener la integridad de la información y la protección de los activos de las Tecnologías de la Información y las Comunicaciones (TIC), se necesita un proceso de administración de la seguridad. Este proceso incluye el establecimiento y mantenimiento de permisos y responsabilidades de seguridad, políticas, estándares y procedimientos. La administración de seguridad también incluye realizar monitoreos de seguridad y pruebas periódicas así como realizar acciones correctivas sobre las debilidades o incidentes de seguridad identificados.

Una efectiva administración de la seguridad protege todos los activos de las TIC para minimizar el impacto causado por vulnerabilidades o incidentes de seguridad (6). Dado el flujo de información que se maneja por las organizaciones en la actualidad, es necesaria una correcta seguridad de la información, la cual permite proteger los recursos de posibles modificaciones y accesos no autorizados.

Los principales objetivos de la seguridad de la información se pueden resumir en:

- ***Confidencialidad***: describe el estado en el cual la información está protegida de revelaciones no autorizadas. (7) (8)
- ***Integridad***: significa que la información no ha sido alterada o destruida por una acción accidental o por un intento malicioso. (7) (8)
- ***Disponibilidad***: referencia al hecho de que una persona autorizada pueda acceder a la información en un apropiado período de tiempo. Las razones de la pérdida de disponibilidad pueden ser ataques o inestabilidades del sistema. (7) (8)
- ***Responsabilidad***: asegurar que las acciones realizadas en el sistema por un usuario se puedan asociar únicamente a ese usuario, que será responsable de sus acciones. Es decir, que un usuario no pueda negar su implicación en una acción que realizó en el sistema. (7) (8)

1.3.1. Autenticación.

Autenticación se define como cualquier método que se utilice para comprobar de manera segura cierta característica de un objeto, como su origen, su no manipulación y su identidad (9).

Existen dos grandes grupos dentro de los métodos de autenticación:

- ***Autenticación de mensaje:*** garantiza la procedencia de un mensaje, identificando al usuario que lo emitió. Por definición, este tipo de autenticación involucra también la integridad de los datos, es decir, permite detectar si el contenido del mensaje fue accidental o maliciosamente alterado.

- ***Autenticación de usuario:*** garantiza que el usuario es quien afirma ser. En este proceso se corrobora la identidad del usuario en tiempo real, es decir, en el mismo momento que se están comunicando el verificador y el usuario.

1.3.1.1. Autenticación de mensaje.

Las técnicas más utilizadas para proveer autenticación de mensaje son (10):

- ***MAC (Message Authentication Code):*** Los códigos de autenticación de mensajes fueron diseñados especialmente para aplicaciones donde se requiere la integridad de los datos (pero no necesariamente su privacidad). Un MAC es un tipo de función unidireccional que toma dos entradas (un mensaje y una clave secreta) y retorna una cadena de tamaño fijo, siendo imposible volver a obtener la misma salida sin conocer la clave. Así, el emisor de un mensaje M, usa la clave secreta compartida K para calcular un MAC *hK (M)* sobre el mensaje y envía al receptor M y *hK (M).* El receptor determina la identidad de la fuente del mensaje (utilizando, por ejemplo, un campo identificador en el texto no cifrado) y separa el MAC de los datos. Luego, calcula un MAC sobre estos datos, utilizando una clave MAC compartida, y compara el MAC calculado con el recibido. Si estos valores coinciden, el receptor interpreta que los datos son auténticos y que no fueron modificados durante su transmisión.
- ***Firma Digital:*** Simula una firma real. Para ello, se genera una prueba digital que sólo el emisor de un mensaje puede crear, pero que todos pueden identificar como perteneciente a ese emisor. Un cifrado utilizando la clave privada del emisor sirve como firma, porque sólo el propietario de la clave privada puede crearlo y todos pueden verificarlo con la clave pública correspondiente. El cifrado (firma) puede aplicarse a todo el mensaje o a un pequeño bloque de datos que sea función del mensaje, por ejemplo, el valor hash. Por tanto, el emisor A envía al receptor B el mensaje M y la firma calculada sobre el hash del mensaje ESKA (H (M)), donde SKA es la clave privada del emisor, E es el algoritmo de cifrado asimétrico utilizado y H (M) es el hash del mensaje. El receptor debe entonces obtener la clave pública del emisor PKA y realizar una operación de cifrado sobre la firma del mensaje, con lo que tendría H´ (M). Posteriormente, B debe calcular el hash del mensaje y compararlo con H´ (M). Si los dos resultados coinciden, se corrobora la autenticidad y la integridad del mensaje.

1.3.1.2. Autenticación de Usuario.

Las técnicas de autenticación de usuario pueden clasificarse en tres categorías, dependiendo de qué deba presentar el usuario para demostrar su identidad (10):

- ***Algo conocido:*** el usuario debe demostrar que conoce algún tipo de información secreta, por ejemplo, una palabra clave, un número de identificación personal (PIN), una clave privada.
- ***Algo que Posee:*** el usuario requiere para identificarse algún tipo de dispositivo, como: una tarjeta magnética, una tarjeta inteligente, un generador de contraseñas o una llave electrónica.
- ***Algo Inherente (Propio del individuo):*** aquí se requiere alguna característica fisiológica del usuario (biometría), por ejemplo: huellas dactilares, la voz, la retina, el iris. Esta categoría puede considerarse como un caso particular de la anterior, donde el dispositivo es el propio usuario.

El subsistema de Administración existente en el GINA, utiliza como técnica de autenticación **algo conocido**, y esta se puede clasificar a su vez en tres categorías de acuerdo al nivel de seguridad que ofrece:

- ***Autenticación Débil:*** es el tipo de autenticación más extendido, y al que los usuarios ya están acostumbrados. Aquí, una contraseña o palabra clave (*password*), asociada a cada usuario, es el secreto compartido entre el usuario y el sistema. Para identificarse, el usuario introduce un identificador (*login*) y su contraseña. El sistema comprueba entonces que esa contraseña coincida con la almacenada para dicho identificador. Ya que las contraseñas son fijas y reutilizables, si un atacante se entera de la clave de un usuario, puede suplantar su identidad fácilmente. Para protegerse de los posibles ataques por fuerza bruta, de los que pueden ser víctima estos sistemas, se limita el número de intentos que el usuario puede hacer desde un terminal concreto y se introducen retardos cuando la contraseña introducida es errónea. (10)
- ***Autenticación Fuerte:*** para este tipo de autenticación una entidad (el demandante) prueba su identidad ante otra entidad (el verificador) demostrando que conoce un secreto asociado con su identidad, pero sin revelar dicho secreto al verificador. Esto se hace respondiendo a un desafío variable. Dicha respuesta es típicamente un número elegido por el verificador (aleatoria y secretamente) al inicio del protocolo. Si un atacante obtiene la respuesta al desafío de un usuario determinado, esa información no le será útil para suplantar posteriormente a ese usuario, ya que el desafío será diferente. Los mecanismos de autenticación fuerte basados en criptografía simétrica, requieren que el demandante y el verificador compartan una clave secreta. Cuando el sistema cuenta

con un gran número de usuarios se suele usar un servidor confiable en línea, con el que cada usuario comparte una clave. Este servidor actúa a manera de Hub2, proporcionando una clave de sesión común a las dos partes comunicantes cada vez que quieran autenticarse una con la otra. El protocolo Kerberos provee autenticación fuerte basada en cifrado simétrico e involucra el uso de una tercera parte de confianza en línea. En los mecanismos de autenticación fuerte basados en criptografía asimétrica, el demandante demuestra que posee su clave privada, bien sea descifrando un desafío cifrado con su clave pública o firmando digitalmente el desafío. Para no comprometer la seguridad del sistema, el par de claves usado por estos mecanismos no debe ser usado con otros propósitos. La Infraestructura de Clave Pública (PKI) utiliza autenticación fuerte basada en criptografía asimétrica para proveer sus servicios de seguridad. (10)

- ***Autenticación de Conocimiento Nulo:*** Los protocolos de conocimientos nulos (**ZK: *Zero-knowledge***) le permiten al probador (el demandante) demostrar el conocimiento de un secreto sin revelar ninguna información útil para el verificador (más allá de la que el verificador es capaz de deducir antes de ejecutar el protocolo). Los sistemas de prueba interactivos son un ejemplo de este tipo de protocolos. Aquí el objetivo del probador es convencer al verificador de que posee algún secreto, contestando correctamente preguntas que requieren del conocimiento de ese secreto para ser respondidas. Estos protocolos emplean técnicas asimétricas pero no utilizan firmas digitales o cifradas con clave pública. Tampoco usan cifradores de bloques, números de secuencia ni sellos de tiempo (*timestamps*). (10)

1.3.1.3. Arquitecturas de Autenticación

Las técnicas de autenticación descritas anteriormente implican algún tipo de clave o secreto compartido entre las partes comunicantes. Debe existir, por tanto, una infraestructura de seguridad que se encargue de la generación, distribución y gestión de los secretos. Esta infraestructura de seguridad brindará una base segura a toda la organización y deberá ser accesible por todas las aplicaciones y objetos de la organización que necesiten seguridad (7). Durante un proceso típico de autenticación, el usuario presenta su credencial (por ejemplo: *login* y *password*) o el resultado de una operación criptográfica que involucre su credencial, a la autoridad de autenticación. Esta valida la credencial usando los datos guardados en la base de datos. Si la credencial proporcionada por el usuario y la guardada en la base de datos coincide, o si el resultado de la operación criptográfica sobre la credencial guardada en la base de datos es igual a la información suministrada por el usuario, la identidad del usuario es considerada auténtica. (8)

Sin embargo, el usuario debe compartir una credencial con cada dominio y autenticarse cada vez que quiera acceder a algún recurso. Dada la diversidad de plataformas de computación, sistemas operativos y de software de control de acceso, lo más deseable es registrarse en múltiples sistemas una vez y simultáneamente a través de una sola transacción. Nace aquí el registro único (SSO – *Single Sign - On*). Con SSO el usuario debe autenticarse sólo una vez para acceder a múltiples sistemas. El registro único puede usarse en infraestructuras con diversas autoridades de autenticación, es decir, implementadas en diferentes plataformas y gobernadas por diversas organizaciones. Las arquitecturas más simples son las que usan un conjunto de credenciales. Las dos más importantes son:

- ***Sistemas Basados en Testigos (Token - Based):*** En una arquitectura basada en testigos, los usuarios obtienen un testigo temporal después de autenticarse exitosamente ante su Trusted Third Party (TTP). Este testigo puede ser guardado en el dispositivo del usuario y reutilizado para probar su identidad ante las TTP de dominios de autenticación secundarios. Para validar el testigo de un usuario, estas TTP usan métodos criptográficos basados en claves secretas establecidas previamente entre ellas y la TTP del dominio de autenticación primario. Estas claves criptográficas permiten establecer una relación de confianza entre diferentes dominios de autenticación. (8)
- ***Sistemas basados en PKI (PKI - Based):*** en esta arquitectura, los usuarios se registran primero ante una autoridad de autenticación confiable, la Autoridad Certificadora (AC). Durante el proceso de registro los usuarios se identifican a través de un conjunto de credenciales. El software del usuario genera un par asimétrico de clave, y la clave pública es ofrecida a la autoridad certificadora para su certificación. Después de recibir las credenciales del usuario y la clave pública, la autoridad certificadora verifica si las credenciales son válidas. Si es así, genera un certificado de clave pública y se lo retorna al usuario. Este certificado y la clave son guardados de manera segura en el dispositivo del usuario u otro medio, como tarjeta inteligente. Estos son usados para probar la identidad del usuario ante otras autoridades de certificación en solicitudes de autenticación posteriores. En esta arquitectura, la relación de confianza entre la autoridad de certificación primaria y las secundarias se establece a través de un certificado expedido por la AC primaria y la AC secundaria. (8)

1.3.2. Autorización

La autorización está estrechamente ligada con la autenticación. Una vez que el usuario ha validado su identidad para acceder a algún recurso, es necesario restringir sus acciones de acuerdo a quien es y que

está tratando de hacer (11). La autorización dota al usuario de privilegios para poder efectuar ciertas operaciones con los datos o recursos protegidos.

1.3.2.1. Modelos de Control de Acceso.

Un factor fundamental para determinar el funcionamiento de todo entorno de autorización es la definición del modelo de control de acceso que se va a establecer. Los mecanismos utilizados para restringir el acceso a los recursos son generalmente de una (o algunas veces la combinación) de dos formas:

- ***Control de Acceso Discrecional (DAC):*** le deja las decisiones de control de acceso al propietario del recurso, de manera que es este quien decide que sujetos pueden realizar determinadas acciones sobre los recursos poseídos. Así, un sujeto con permisos de acceso puede otorgarlos a otro sujeto. (12)
- ***Control de Acceso Obligatorio (MAC):*** este control de acceso se basa en reglas establecidas por una autoridad central. MAC restringe el acceso a los objetos basándose en la sensibilidad de la información que estos contienen (representada por una etiqueta) y en la autorización formal de los sujetos para acceder a dicha información. Los objetos son considerados como entidades pasivas que almacenan información, mientras que los sujetos son entidades activas que realizan peticiones de acceso a los objetos. (12)

Existen diversos modelos de control de acceso de tipo DAC y/o MAC. Los más comunes son:

- ***Control de Acceso Basado en Identidad (IBAC):*** en este caso los permisos de acceso se asocian directamente al identificador del sujeto, es decir, el nombre del usuario. Por tanto se garantiza el acceso al recurso sólo cuando exista dicha asociación. Con IBAC no se asocian etiquetas de seguridad a los usuarios, por lo que este es principalmente un mecanismo de control de acceso discrecional. Un ejemplo de IBAC son las Listas de Control de Acceso (ACLs), encontradas comúnmente en sistemas operativos y servicios de seguridad en red. Una Lista de Control de Acceso contiene los identificadores de los usuarios junto con sus derechos de acceso a un recurso determinado, como leer, escribir, ejecutar. Esta estructura básica de autorización únicamente extiende el concepto de autenticación, ya que si el usuario no puede autenticarse correctamente ante el guardián del recurso, su solicitud de acceso es denegada. Entre más usuarios soliciten el acceso a un recurso más identificadores contendrá la ACL, lo que dificulta el manejo de estas listas y las hace una alternativa poco escalable. Por otra parte, la decisión de control de acceso no

depende de alguna función o característica de la organización a la que pertenece el usuario sino solamente de los identificadores, así que su uso es inapropiado a nivel empresarial. (13)

- ***Control de Acceso Basado en Roles (RBAC):*** restringe el acceso a los recursos basándose en la función o rol que desempeña el sujeto dentro de la organización. Los permisos para acceder a un recurso son asignados a cada rol, en lugar de asociarlos directamente al identificador del sujeto. El RBAC es escalable y reduce significativamente la cantidad de información de administración necesaria, ya que los permisos no son asignados constante e individualmente a cada usuario. RBAC es primordialmente un mecanismo de control de acceso discrecional. Generalmente no tiene en cuenta las características de los recursos (más que sus identificadores) y no obtienen ninguna información relevante de seguridad del entorno. Este precisamente es el modelo de control de acceso utilizado en el sistema de administración de GINA y por consiguiente a continuar usando en la solución actual. (13)
- ***Control de Acceso Basado en Atributos (ABAC):*** en ABAC los privilegios son establecidos en base a la colección de atributos que posee el usuario y una política que los determina. ABAC es la convergencia natural de los modelos de control de acceso IBAC y RBAC. La representación de las políticas en ABAC es semánticamente más rica y expresiva. Además posee una mayor granularidad ya que puede basarse en cualquier combinación de atributos de sujeto, de recursos y de entorno. (13)
- ***Control de Acceso Basado en Mallas (LBAC):*** en este caso una colección totalmente ordenada de etiquetas de seguridad se combina con un grupo de categorías y forman una *malla*. Con ellos se obtienen un conjunto de clases de seguridad que varía desde la más baja a la más alta. Generalmente el modelo LBAC es manejable cuando existe un número relativamente pequeño de etiquetas de seguridad y categorías, por lo sólo es efectivo para ciertos escenarios de seguridad poco granulados y carentes de flexibilidad y escalabilidad. LBAC es un mecanismo de control de acceso obligatorio. (13) (14)

1.4. Ventanilla Única de Comercio Exterior.

La Ventanilla Única de Comercio Exterior (VUCE) es un sistema integrado que permite a las partes involucradas en el comercio exterior y transporte internacional gestionar a través de medios electrónicos y por un solo punto, los trámites requeridos por las entidades de control competentes para el tránsito, ingreso o salida del territorio nacional de mercancías (15).

1.5. Estado del Arte.

El presente trabajo es precisamente una ampliación de los subsistemas ya existentes en el sistema GINA; específicamente de los subsistemas RC y de Administración (suAdmin), los cuales serán extendidos para lograr las nuevas especificaciones requeridas por el sistema.

El subsistema de Administración, está implementando bajo las condiciones y métricas del proyecto Aduana, es una excelente solución para la gestión de la seguridad del sistema y actualmente se encuentra en uso con magníficos resultados en entorno de producción; pero su diseño actual no permite la gestión de la seguridad de los trabajadores externos de la AGR, por lo que necesita de cambios en su diseño y estructura. En el caso del subsistema de RC, ante el surgimiento de nuevas especificaciones se hace igualmente necesaria una restructuración de su modelo de datos, lo que genera varios cambios en su implementación. Además, se hace necesaria la implementación de nuevas funcionalidades en el mismo. A pesar de esto, el subsistema está implementado siguiendo excelentes patrones y diseños, lo que lo hace una muy buena solución.

Teniendo en cuenta lo anteriormente mencionado, se hace solamente un análisis superficial de los sistemas ya existentes, puesto que el objetivo no es el cambio de flujos y métodos actualmente en uso, sino el enriquecimiento y mejoramiento de los mismos con nuevas experiencias que pueden de una u otra forma dotar al sistema de mayor eficiencia y confiabilidad. Para ello se divide el análisis en dos etapas, primeramente el estudio de sistemas que gestionan de alguna forma un RC y posteriormente los que gestionan seguridad y control acceso; en ambos casos se hace énfasis en los desarrollados en la UCI.

1.5.1. Sistemas de Gestión de Registros Centrales.

CEDRUX

El proyecto CEDRUX desarrollado en la UCI, es una alternativa cubana para la gestión de recursos empresariales, o sea, un sistema ERP (*Enterprise Resourse Planning*, o Planificación de Recursos Empresariales en español). Posee un componente con funcionalidades análogas a las de un RC, el cual se incluye en el subsistema Configuración; en el mismo se recogen datos de la entidad como si la misma es cliente o proveedor, con la idea de que una misma entidad puede fungir como cliente ante una empresa y como proveedor ante otra, se establecen además relaciones entre dichas entidades según el tipo que sean, las tablas donde se guardan estos datos utiliza nomencladores, semejante a las tablas de control en el sistema GINA. Este componente realiza de una forma u otra las mismas funcionalidades del RCE de la

AGR, pero no se ajusta a las necesidades específicas de la institución así como las herramientas y arquitectura base no son del todo compatibles e integrables con los sistemas usados en el desarrollo del producto GINA. Además para su uso habría que ubicar su contenido en el directorio público de la aplicación, elemento que dota al sistema de un alto nivel de inseguridad, al proporcionar su implementación a un posible atacante.

SAGEB

El Sistema Automatizado de Gestión Bancaria (SAGEB) es un sistema que se está desarrollando en la UCI, este cuenta con un subsistema para el manejo de la información de entidades naturales o jurídicas acorde a las necesidades y funcionalidades que en este se generan; este subsistema cuenta con dos módulo para dicha operación: Personas Naturales y Personas Jurídicas, los cuales básicamente utilizan dos tablas en la base de datos para su funcionamiento: *c_client y c_person.* Como herramientas para su desarrollo se utiliza Microsoft SQL Server como gestor de bases de datos, el cual no cuenta con relaciones entre sus tablas y las interacciones con el subsistema se realizan mediante procedimientos almacenados; Hibernate para la capa de acceso a datos, Java para el desarrollo de la lógica del negocio, y Dojo para la interfaz de usuario. Como se puede apreciar, además de tener un componente que vanamente realiza las tareas que la AGR requiere, este posee una tecnología y arquitectura totalmente diferente a la empleada en el GINA, y su utilización traería consigo grandes complicaciones de integración entre los sistemas.

SIDUNEA

El Sistema Computarizado de Aduanas (SIDUNEA) es una herramienta informática que se ha introducido en varios países, con el objeto de mejorar el comercio internacional, a través de medidas que incluyen reformas a la práctica administrativa ya existente. Fue diseñado para proveer estadísticas de comercio. En la actualidad el sistema SIDUNEA se encuentra operando o está en proceso de implementación, en más de 80 países alrededor del mundo. Es un sistema abierto que utiliza Java en su negocio, y Oracle para el almacenamiento de los datos. El sistema SIDUNEA contiene ocho módulos, a continuación se describen algunas funcionalidades de aquellos que, de una forma u otra, persiguen objetivos similares a los procesos del RC. (3)

MODSEL o Módulo de Selectividad**:** Permite a la Aduana controlar la selección y el flujo de las declaraciones a través del sistema de procesamiento de la declaración de aduana. Contiene los controles

para bloquear la liquidación de las declaraciones seleccionadas y tiene muchas funciones para hacer consultas e imprimir reportes.

MODCBR**:** Es el módulo central del sistema, la base de los procedimientos de presentación, control, cobro y liquidación de los impuestos. Trabaja principalmente con declaraciones de mercancías, su ingreso al sistema, su verificación local y remota, registro, aplicación del resultado de selectividad y validación.

El sistema SIDUNEA emplea códigos internacionales y estándares desarrollados por la Organización Internacional para la Estandarización (ISO), por la Organización Mundial de Aduanas (OMA) y por la Organización de Naciones Unidas (ONU). El mismo fue el primer sistema computarizado que se instaló en la AGR pero no todos los módulos se pudieron implantar por incongruencias en su concepción con las características específicas de la misma. (1)

SUA

El sistema SUA junto al DataEase 4.0, forma parte del RC actualmente. Todos sus módulos validan y controlan las entradas de datos con nomencladores y clasificadores (aproximadamente 100) que oportunamente fueron diseñados, los cuales facilitan la flexibilidad del sistema, además de tener organizada la información y así asegurar la consistencia de los datos. Este módulo se concibió con el propósito de relacionar las diferentes informaciones de carácter constante que serán usadas en el sistema, además de organizar estas informaciones. La estructura y contenido de los nomencladores y clasificadores están en correspondencia con las necesidades de análisis, agrupamiento o des agrupamiento de información que se requiera durante el procesamiento automatizado. Los datos son introducidos con posibilidad de ser modificados, es válido aclarar que en el caso de las modificaciones y eliminaciones se realizan de forma lógica actualizando sólo la fecha de vencimiento del artículo (tomado como experiencia del SIDUNEA en aquel momento), esto ocurre en todas las tablas ya que por requerimientos del sistema, la información se guarda durante cinco años y para garantizar la consistencia, se realizan las actualizaciones de esta manera (16).

Actualmente este sistema no cumple con las necesidades y requerimientos que surgen en la AGR tras el aumento de la información y la automatización de procesos, elementos que son expuestos en la problemática del presente trabajo.

1.5.2. Sistemas de Control de Acceso

Sistema de Autenticación de Aplicaciones de la Intranet

La investigación en la UCI sobre los software de gestión de seguridad arrojó como resultado la existencia del Sistema de Gestión de Sesiones basado en la arquitectura SSO cuyo objetivo es gestionar las sesiones de los usuarios en un único proceso de autenticación invisible a los ojos de los mismos, a través de un sistema con una fachada de servicios web que sea capaz de gestionar la apertura y cierre de sesiones por parte de las personas que trabajan en el dominio uci.cu. Este sistema no soluciona el problema existente, ya que sólo propone centralizar la autenticación de usuarios y mantiene la autorización como responsabilidad de cada aplicación, elemento que en la solución buscada no es admisible, pues lo requerido es un sistema de administración centralizado.

Sistemas de Gestión de Acceso

En el año 2008 se realiza un trabajo de diploma con el nombre “Sistema de Gestión de Accesos (SGA)”, basado en la necesidad de implementar un sistema que controle centralizadamente los accesos y garantice seguridad en la red y aplicaciones de la misma, el cual brinda principalmente, los servicios de autenticación y autorización de usuarios a las disímiles aplicaciones. Asimismo soporta los mecanismos necesarios para, a través de él, obtener información de las aplicaciones, funcionalidades, roles, usuarios y grupos de usuarios de la UCI. Este sistema tiene la desventaja de que los usuarios tienen que autenticarse cada vez que acceden a una aplicación, elemento que lo hace ineficiente en entornos que necesiten de entrada y salida constante de varios sistemas.

Sistema de Gestión de Sesiones

En el 2008 se realiza el trabajo de diploma “Sistema de Gestión de Sesiones (SGS)” el cual se convierte en una iniciativa viable de protección y control que permite ofrecer servicios de valor añadido con altos niveles de seguridad y confianza a los usuarios. La implantación de este sistema, logra mejorar la seguridad de la red en la UCI, permitiendo a su vez a los usuarios disminuir su trabajo de autenticación y el tiempo que pierden los mismos en este proceso. Además en la UCI cada uno de los servicios o aplicaciones cuenta con su propio componente de seguridad, lo cual generalmente compromete la seguridad de todo el sistema, dado que el nivel de seguridad de todo un sistema es igual al nivel de seguridad del componente más inseguro que tenga. Esta es una de las razones por las que no se usa esta solución de software, a pesar de permitir a los usuarios autenticarse una única vez y hacer un control de sesiones eficiente.

Acaxia

Por otra parte se encuentra el Sistema de Seguridad “Acaxia”, anteriormente conocido como Sistema de Gestión Integral de Seguridad (SIGIS), desarrollado por el proyecto ERP – Cuba. Posee módulos para la gestión de usuarios, roles y grupos de usuarios, además de que autenticación mediante LDAP totalmente configurable. Es un componente que cubre muchas necesidades de seguridad pero que no se acopla a la arquitectura de Symfony. Está implementado con Zend Framework y para su implantación requiere que se incluya el componente completo dentro del directorio público del proyecto de Symfony y este caso sería un riesgo para la seguridad del sistema al hacer posible el acceso de un posible atacante a la codificación de la herramienta.

1.6. Ingeniería de Requerimientos

En el glosario de Terminología de Ingeniería de Software de la IEEE se define requerimiento como: *“Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.*” (17)

“Una condición o capacidad que debe estar presente en un sistema o componente de un sistema para satisfacer un contrato, estándar, especificación u otro documento formal.” (17)

La Ingeniería de Requerimientos (IR) cumple un papel primordial en el proceso de producción de software, ya que se enfoca en un área fundamental: definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedad, en forma consistente y compacta, las necesidades de los usuarios o clientes; de esta manera, se pretende minimizar los problemas relacionados por la mala gestión de requisitos en el desarrollo de sistemas. (18)

Otros de los conceptos de la ingeniería de requerimientos son propuestos por Pressman el cual plantea. *“Ingeniería de Requerimientos ayuda a los ingenieros de software a entender mejor el problema en cuya solución trabajan. Incluye el conjunto de tareas que conducen a comprender cuál será el impacto del software sobre el negocio, qué es lo que el cliente quiere y cómo interactúan los usuarios finales con el software.”* (19)

1.6.1. Actividades de la Ingeniería de Requerimientos

Para realizar este proceso de Ingeniería de Requerimientos, no existe una única técnica estandarizada y estructurada que ofrezca un marco de desarrollo que garantice la calidad del resultado. Existe en cambio un conjunto de técnicas, cuyo uso proponen las diferentes metodologías para el desarrollo de aplicaciones web. Se debe tener en cuenta que la selección de las técnicas y el éxito de los resultados que se

obtengan, depende en gran medida tanto del equipo de análisis y desarrollo, como de los propios clientes o usuarios que en ellas participen. Este se puede dividir en tres grandes actividades:

Captura de requisitos**:** La captura de requisitos es la actividad mediante la que el equipo de desarrollo de un sistema de software extrae, de cualquier fuente de información disponible, las necesidades que debe cubrir dicho sistema. El proceso de captura de requisitos puede resultar complejo, principalmente si el entorno de trabajo es desconocido para el equipo de analistas, y depende mucho de las personas que participen en él. Por la complejidad que todo esto puede implicar, la ingeniería de requisitos ha trabajado desde hace años en desarrollar técnicas que permitan hacer este proceso de una forma más eficiente y precisa. (20)

Especificación de Requerimientos**:** En esta fase se documentan los requerimientos acordados con el cliente, en un nivel aprobado de detalles. En la práctica esta actividad se va desarrollando en conjunto con el análisis debido a que en esta se plasman en documentos los requerimientos del cliente.

Validación de Requerimientos**:** Los requisitos una vez definidos necesitan ser validados. La validación de requisitos tiene como misión demostrar que la definición de los requisitos define realmente el sistema que el usuario necesita o el cliente desea. Es necesario asegurar que el análisis realizado y los resultados obtenidos de la etapa de definición de requisitos son correctos. Pocas son las propuestas existentes que ofrecen técnicas para la realización de la validación y muchas de ellas consisten en revisar los modelos obtenidos en la definición de requisitos con el usuario para detectar errores o inconsistencias.

1.6.2. Técnicas de captura de requerimientos

Entrevistas: resultan una técnica muy aceptada dentro de la ingeniería de requisitos y su uso está ampliamente extendido. Las entrevistas le permiten al analista tomar conocimiento del problema y comprender los objetivos de la solución buscada. A través de esta técnica el equipo de trabajo se acerca al problema de una forma natural.

Básicamente, la estructura de la entrevista abarca tres pasos: identificación de los entrevistados, preparación de la entrevista, realización de la entrevista y documentación de los resultados (protocolo de la entrevista). A pesar de que las entrevistas son esenciales en el proceso de la captura de requisitos y con su aplicación el equipo de desarrollo puede obtener una amplia visión del trabajo y las necesidades del usuario, es necesario destacar que no es una técnica sencilla de aplicar. Requiere que el entrevistador

sea experimentado y tenga capacidad para elegir bien a los entrevistados y obtener de ellos toda la información posible en un período de tiempo siempre limitado. (21)

Tormenta de ideas**:** es una técnica de reuniones en grupo cuyo objetivo es que los participantes muestren sus ideas de forma libre. Consiste en la mera acumulación de ideas y/o información sin evaluar las mismas. El grupo de personas que participa en estas reuniones no debe ser muy numeroso (máximo 10 personas), una de ellas debe asumir el rol de moderador de la sesión, pero sin carácter de controlador. Como técnica de captura de requisitos es sencilla de usar y de aplicar, puesto que no requiere tanto trabajo en grupo. Además suele ofrecer una visión general de las necesidades del sistema, pero normalmente no sirve para obtener detalles concretos del mismo, por lo que suele aplicarse en los primeros encuentros. (21)

Mapa Conceptual**:** son grafos en los que los vértices representan conceptos y las aristas representan posibles relaciones entre dichos conceptos. Estos grafos de relaciones se desarrollan con el usuario y sirven para aclarar los conceptos relacionados con el sistema a desarrollar. Son muy usados dentro de la ingeniería de requisitos, pues son fáciles de entender por el usuario, más aún si el equipo de desarrollo hace el esfuerzo de elaborarlo en el lenguaje de éste. (21)

Casos de Uso**:** aunque inicialmente se desarrollaron como técnica para la definición de requisitos, algunos autores proponen casos de uso como técnica para la captura de requisitos. Los casos de uso permiten mostrar el contorno (actores) y el alcance (requisitos funcionales expresados como casos de uso) de un sistema. Un caso de uso describe la secuencia de interacciones que se producen entre el sistema y los actores del mismo para realizar una determinada función. Los actores son elementos externos (personas, otros sistemas, entre otros) que interactúan con el sistema como si de una caja negra se tratase. Un actor puede participar en varios casos de uso y un caso de uso puede interactuar con varios actores. La ventaja esencial de los casos de uso es que resultan muy fáciles de entender para el usuario o cliente, sin embargo carecen de la precisión necesaria si no se acompañan con una información textual o detallada con otra técnica como pueden ser los diagramas de actividades. (21)

Existen más técnicas para la captura de requerimientos (análisis de otros sistemas, estudio de la documentación, entre otras), incluso también es común encontrar alternativas que combinen varias de estas técnicas. Sin embargo, las presentadas ofrecen un conjunto representativo de las más utilizadas.

1.6.3. Técnicas de validación de requerimientos

Revisiones de requerimientos**:** los requerimientos son analizados sistemáticamente por un equipo de revisores. Una revisión de requerimientos es un proceso manual que involucra a personas tanto de la organización del cliente como de la contratista. Las revisiones de requerimientos pueden ser informales o formales. (22)

Auditorías**:** la revisión de la documentación con esta técnica consiste en un chequeo de los resultados contra una Checklist predefinida o definida a comienzos del proceso, es decir sólo una muestra es revisada. (21)

Prototipos**:** algunas propuestas se basan en obtener de la definición de requisitos prototipos que, sin tener la totalidad de la funcionalidad del sistema, permitan al usuario hacerse una idea de la estructura de la interfaz del sistema con el usuario. Esta técnica tiene el problema de que el usuario debe entender que lo que está viendo es un prototipo y no el sistema final. (21)

Generación de casos de prueba**:** los requerimientos deben poder probarse. Si las pruebas para éstos se conciben como parte del proceso de validación, a menudo revela los problemas en los requerimientos. Si una prueba es difícil o imposible de diseñar, normalmente significa que los requerimientos serán difíciles de implementar y deberían ser considerados nuevamente. Desarrollar pruebas para los requerimientos del usuario antes que se escriba código es una parte fundamental de la programación extrema. (22)

En el presente trabajo se hace uso de las técnicas, **revisión del documento de requerimientos** y el uso de **prototipos**, elemento que será abordado más adelante en el presente trabajo.

1.7. Diseño de Software

En el diseño se modela el sistema y se encuentra la forma para que soporte todos los requerimientos, incluyendo los requisitos no funcionales y otras restricciones, se asienta en el núcleo técnico del proceso de ingeniería del software y se aplica independientemente del paradigma de desarrollo utilizado. En la elaboración de la solución es preciso tener presente ciertos estándares, estilos y patrones que son muy útiles en la obtención de un diseño de software robusto.

1.7.1. Metodologías de Diseño

Metodología de Diseño se refiere al desarrollo de un sistema o método para una situación única. Hoy en día, el término suele aplicarse a los campos tecnológicos en referencia al diseño web, software o diseño

de sistemas de información. A continuación de describen algunas de estas tecnologías que son usadas en el desarrollo de la solución.

Diseño Orientado al Uso: Cuando el diseño se centra en las tareas asociadas al uso y sus objetivos. Este enfoque penaliza la usabilidad, ya que incrementa la curva de aprendizaje, pero logra solventar problemas complejos o de alta criticidad.

Diseño Centrado en el Usuario: (UCD *User Center Design*) es la metodología de diseño más habitual en el mundo. Este enfoque coloca todas las necesidades, deseos y limitaciones del usuario como núcleo del proceso de diseño. Por lo cual esta metodología conlleva por definición investigación y análisis del usuario. Así mismo implica que el proyecto debe ser distribuido por faces para garantizar los resultados. Dentro del UCD hay metodologías específicas, como son KES (*Kansei Engineering System*) o QFD (*Quality Function Deployment*). (23)

Enfoque Ascendente: se refiere a la identificación de aquellos procesos que necesitan computarizarse conforme vayan apareciendo, su análisis como sistemas y su codificación; o bien, la adquisición de paquetes de software para satisfacer el problema inmediato.

Enfoque Descendente: Consiste en representar una imagen del sistema y luego dividirla en fracciones más pequeñas. El diseño descendente es compatible con la manera de pensar sobre los sistemas en general.

Desarrollo Modular: Su funcionamiento se basa en una descomposición de la programación en fracciones lógicas y manejables. Este tipo de programación se apega bien al diseño descendente porque enfatiza las interfaces entre los módulos, más que mantenerlas ignoradas hasta el final del desarrollo del sistema. Cada módulo debe ser funcionalmente cohesivo, de manera que satisfaga sólo una función (23).

1.7.2. Arquitectura de Software

En el desarrollo de aplicaciones informáticas la arquitectura de software es un término muy frecuente y de gran referencia ya que sirve para guiar la implementación y el desarrollo del sistema desde etapas tempranas del diseño. Existen muchas definiciones acerca del tema, y aunque todas son consideraciones especializadas de distintos autores, coinciden en que a grandes rasgos se refiere a la estructura del sistema, constituida por componentes de software y relaciones entre estos. (24)

Algunas definiciones formales han sido expuestas como por ejemplo. *"La arquitectura del software de un programa o sistema de computación es la estructura o estructuras del sistema que comprende los*

elementos del software, las propiedades externamente visibles de esos elementos, y las relaciones entre ellos". (24)

"La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones". (25)

Esta definición es un poco amplia, por lo que una de las más aplicadas es la establecida por la IEEE STD 1471-2000 (*Software Engineering Standards Committee of the IEEE Computer Society, 2000*): *"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución."*

1.7.3. Estilos Arquitectónicos

Los estilos arquitectónicos no son más que patrones pertenecientes a un tipo de vista concreto, que definen una serie de restricciones a los tipos de elementos y relaciones de la vista arquitectónica. Algunos estilos son universales, mientras que otros definen un tipo particular de software. En cualquier caso, la arquitectura de un sistema está compuesta por vistas pertenecientes a múltiples estilos. Brevemente descritos, los estilos conjugan elementos, conectores, configuraciones y restricciones. Al estipular los conectores como elemento de juicio de primera clase, el concepto de estilo, incidentalmente, se sitúa en un orden de discurso y de método que el modelado orientado a objetos en general y UML (*del inglés, Unified Modeling Language*) en particular no cubren satisfactoriamente. La descripción de un estilo se puede formular en lenguaje natural o en diagramas, pero lo mejor es hacerlo en un lenguaje de descripción arquitectónica o en lenguajes formales de especificación.

A diferencia de los patrones de diseño, que son centenares, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo. Es digno de señalarse el empeño por incluir todas las formas existentes de aplicaciones en un conjunto de dimensiones tan modestas. Las arquitecturas complejas o compuestas resultan del agregado o la composición de estilos más básicos.

Estilos de Flujo de Datos

- Tubería y filtros.
- Secuencial en lote.

Estilos Centrados en Datos

- Arquitecturas de Pizarra o Repositorio.
- Bases de Datos.
- Sistemas de Hipertextos.

Estilos de Llamada y Retorno

- Model-View-Controller(MVC).
- Arquitecturas en Capas.
- Arquitecturas Orientadas a Objetos.
- Arquitecturas Basadas en Componentes.

Estilos Derivados

- C2
- GenVoca
- REST

Estilos de Código Móvil

- Arquitectura de Máquinas Virtuales.

Estilos heterogéneos

- Sistemas de control de procesos.
- Arquitecturas Basadas en Atributos.

Estilos Peer-to-Peer

- Arquitecturas Basadas en Eventos.
- Arquitecturas Orientadas a Servicios (SOA).
- Arquitecturas Basadas en Recursos.

1.7.4. Patrones

Cada patrón describe un problema que ocurre y una y otra vez en un entorno determinado y describe también el núcleo de la solución del problema, de forma que puede utilizarse un millón de veces sin tener que hacer dos veces lo mismo. Durante el desarrollo de software suelen surgir problemas comunes, los

cuales son resueltos con el uso de alternativas de soluciones que pueden ser muy efectivas, pues en la práctica y en el transcurrir de los años se ha demostrado que son eficientes frente a dificultades presentadas, estas variantes son como esquemas de situaciones que se presentan con frecuencia, y traen asociada la manera en que se solventan, estos elementos son llamados patrones. Dentro de las definiciones de patrones se encuentra la propuesta por Craig Larman: *"Un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos sobre cómo aplicarlo en nuevas situaciones, o sea, un patrón es una descripción de un problema bien conocido que suele incluir: descripción, escenario de uso, solución concreta, consecuencias de utilizar el patrón, ejemplos de implementación y lista de patrones relacionados."*

¿Qué son los patrones?

- Solucionan un problema: los patrones capturan soluciones, no sólo principios o estrategias abstractas.
- Son un concepto probado: capturan soluciones demostradas, no teorías o especulaciones.
- La solución no es obvia: los mejores patrones generan una solución a un problema de forma indirecta.
- Describen participantes y relaciones entre ellos: describen módulos, estructuras del sistema y mecanismos complejos.
- El patrón tiene un componente humano significativo: todo software proporciona a los seres humanos confort y calidad de vida (estética y utilidad).

Patrones de Arquitectura

Los patrones de arquitectura están relacionados fundamentalmente con la estructura de un sistema de software. Especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes. Describen un problema particular y recurrente del diseño, que surge en un contexto específico, y presenta un esquema genérico y probado de su solución. (26)

Patrón Modelo Vista Controlador (MVC) Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres capas o componentes distintos. El patrón MVC se utiliza frecuentemente en aplicaciones web, donde la vista es la página HTML (*del inglés, Hypertext Markup Language*) que se visualiza en el navegador y el código que proporciona de datos dinámicos a la página, el modelo es el Sistema de Gestión de Base de Datos y la lógica de negocio y el

controlador es el responsable de recibir los eventos de entrada desde la vista, enviarlos al modelo y manejar también las respuestas del modelo y transmitirlas hacia la vista. (27)

Modelo: El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe procesar. El modelo desconoce la existencia de las vistas y del controlador. Ese enfoque suena interesante, pero en la práctica no es aplicable pues deben existir interfaces que permitan a los módulos comunicarse entre sí. Esta es la recepción específica del dominio de la información sobre la cual funciona la aplicación. El modelo es una forma de llamar la capa de dominio. La lógica de dominio añade significados a los datos.

Vista**:** Las vistas son el conjunto de clases que se encargan de mostrar al usuario la información contenida en el modelo. Una vista está asociada a un modelo, pudiendo existir varias vistas asociadas al mismo modelo. Una vista obtiene del modelo solamente la información que necesita para desplegar y se actualiza cada vez que el modelo del dominio cambia por medio de notificaciones generadas por el modelo de la aplicación.

Controlador**:** El controlador es un objeto que se encarga de dirigir el flujo del control de la aplicación debido a mensajes externos, como datos introducidos por el usuario u opciones del menú seleccionadas por él. A partir de estos mensajes, el controlador se encarga de modificar el modelo o de abrir y cerrar vistas. El controlador tiene acceso al modelo y a las vistas, pero las vistas y el modelo no conocen de la existencia del controlador.

Patrones de Diseño

Los Patrones de Diseño no son más que soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan, son descripciones de clases cuyas instancias colaboran entre sí y brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares.

Patrones GRASP GRASP es un acrónimo que significa General Responsibility Assignment Software Patterns (*en español, patrones generales de software para asignar responsabilidades*). El nombre se eligió para indicar la importancia de captar *(grasping)* estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Tienen como motivación:

- La asignación de responsabilidades como la habilidad más importante en el análisis y diseño orientado por objetos.
- Respetar los principios fundamentales como uno de los factores críticos, para obtener diseños reutilizables, mantenibles y extendibles.

Patrones GoF El grupo de GoF (*del inglés, Gang of Four*) clasificaron los patrones en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento.

- Creacionales: los patrones creacionales se relacionan con las formas de crear instancias de objetos. El objetivo de estos patrones es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados.
- Estructurales: los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos.
- Comportamiento: los patrones de comportamiento ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre objetos.

1.8. Metodologías, lenguajes y herramientas empleadas.

En la actualidad el uso de aplicaciones web representa un elemento que ha contribuido considerablemente al desarrollo de numerosas empresas. Este tipo de aplicaciones presenta ventajas significativas en cuanto al ahorro de tiempo de los procesos, compatibilidad y disponibilidad de la información. Durante estudios previos al desarrollo del sistema GINA se definió la línea base de la arquitectura la cual se compone por las principales tecnologías o metodologías empleadas en el desarrollo del sistema. A continuación se muestra una descripción de algunas de estas haciendo énfasis en sus principales características.

1.8.1. Metodología de desarrollo

Las Metodologías de Desarrollo de Software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto software. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo software, pero los

requisitos de un software a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del software.

En el Departamento de Soluciones para la Aduana, del centro CEIGE, se propone un modelo de desarrollo que se nutre de varios años de trabajo y busca agilizar y simplificar el proceso de desarrollo sin perder calidad y eficiencia en el proceso. Este modelo, que se titula "Modelo de Desarrollo de Software del Departamento de Soluciones para la Aduana" (28), se utiliza como quía rectora para llevar a cabo sus proyectos, dicha definición incluye los flujos de trabajo, encuentros y reuniones básicas, los roles y sus responsabilidades así como la interacción entre ellos. Las plantillas están regidas por el programa de mejoras para alcanzar el nivel 2 en CMMI[4]. El modelado se realiza por Procesos, escritos en notación BPMN (*Business Process Modeling Notation*) para así lograr una comprensión fácil para todos los usuarios, tanto del negocio, como para los analistas implicados y los que realizan el diseño de la solución. BPMN crea un puente estandarizado entre los procesos de negocio, diseño e implementación. Este modelo dirige las actividades que se realizan por rol, les da una responsabilidad, y define qué tareas deberían ser desarrolladas. Define además un ciclo de vida para los proyectos del departamento el cual se muestra a continuación:

[4] CMMI: Modelo de calidad del software que clasifica las empresas en niveles de madurez.

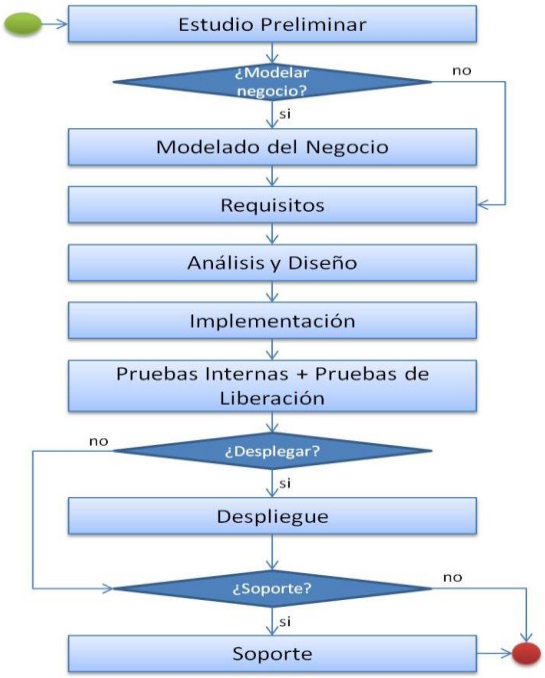

Figura No 1. Ciclo de Vida de los proyectos del Departamento de Soluciones para la Aduana

Los elementos característicos de este modelo son:

- Roles y responsabilidades
- Actividades
- Descripción de proceso
- Artefactos opcionales

Para más información sobre el modelo de desarrollo de software utilizado por el Departamento de Soluciones para la Aduana, dirigirse al documento que lleva por título: “Modelo de Desarrollo de Software”. (28)

1.8.2. Lenguajes y Herramientas de Modelado y Diseño

En la actualidad muchas empresas se han extendido a la adquisición de herramientas CASE[5], con el fin de automatizar los aspectos clave de todo el proceso de desarrollo de un sistema y así incrementar su posición en el mercado competitivo. Algunas de estas herramientas tienen un valor económico muy alto y requieren costos de entrenamiento de personal muy altos, además se enfrenta la falta de adaptación de la

[5] **CASE:** siglas en ingles que se utilizan para referirse a Ingeniería de Software Asistida por Computadora

herramienta, a la arquitectura de la información en la que está compuesta y a las metodologías de desarrollo utilizadas por la organización.

1.8.2.1. Lenguaje de modelado UML 2.0

El lenguaje Unificado de Modelado (UML) es un lenguaje de modelado de sistemas de software y es en la actualidad el más conocido y utilizado. Su función es la de visualizar, especificar, construir y documentar un sistema. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir, en otras palabras para describir el modelo. Es válido decir que UML no está concebido para la programación estructurada puesto que sólo se diagrama la realidad a partir de requerimientos.

Diagramas de Estructura:

- Diagrama de clases.
- Diagrama de componentes.
- Diagrama de objetos.
- Diagrama de estructura compuesta.
- Diagrama de despliegue.
- Diagrama de paquetes.

Diagramas de Comportamiento:

- Diagrama de actividades.
- Diagrama de casos de uso.
- Diagrama de estados.

Diagramas de Interacción:

- Diagrama de secuencia.
- Diagrama de comunicación.
- Diagramas de tiempos.
- Diagrama de vista de interacción.

1.8.2.2. Visual Paradigm for UML

Visual Paradigm for UML (VP-UML) es un modelador UML que permite el diseño de sistemas con todo tipo de tipos de diagramas UML. También es utilizado para diseñar diagramas de casos de uso, diagramas de requerimiento y diseño de bases de datos relacionales. Con VP-UML, el equipo de desarrollo de software puede realizar el análisis y diseño de sistemas con eficacia(28). Es una herramienta

multiplataforma y se integra con algunas herramientas realizadas en Java entre las cuales se puede citar el NetBeans IDE. Debido a las características y facilidades argumentadas anteriormente se elige la herramienta Visual Paradigm for UML en su versión 3.4 para la realización del diseño de la solución.

1.8.3. Lenguajes de programación

Un lenguaje de programación no es más que un conjunto de sintaxis y reglas semánticas que definen los programas del computador. Es una técnica estándar de comunicación para entregarle instrucciones al computador. Un lenguaje le da la capacidad al programador de especificarle al computador, qué tipo de datos actúan y que acciones tomar bajo una variada gama de circunstancias, utilizando un lenguaje relativamente próximo al lenguaje humano. Un programa escrito en un lenguaje de programación necesita pasar por un proceso de compilación, interpretación o intermedio, es decir, ser traducido al lenguaje de máquina para que pueda ser ejecutado por el ordenador.

En la actualidad son muy variados los lenguajes de programación que se usan para desarrollar las distintas soluciones, partiendo desde aquellos privativos por los cuales es necesario pagar una patente y llegando hasta los lenguajes de carácter libre cuyo soporte es dado por la comunidad de usuarios que deseen colaborar.

1.8.3.1. Lenguajes utilizados en el lado del cliente.

Para realizar la programación en el lado del cliente[6] fueron elegidos los siguientes lenguajes:

XHTML:

El lenguaje HTML es un estándar reconocido en todo el mundo y cuyas normas define un organismo sin ánimo de lucro llamado W3C (*World Wide Web Consortium*). Como se trata de un estándar reconocido por todas las empresas relacionadas con el mundo de Internet, una misma página HTML se visualiza de la misma manera en cualquier navegador de cualquier sistema operativo. (29)

El lenguaje XHTML es muy similar al lenguaje HTML. De hecho, XHTML no es más que una adaptación de HTML al lenguaje XML. Técnicamente, HTML es descendiente directo del lenguaje SGML[7] (*Standard Generalized Markup Language*), mientras que XHTML lo es del XML (que a su vez, también es descendiente de SGML). (29)

[6] **Lenguajes en el lado del cliente:** son los lenguajes que basan su procesamiento en el cliente web, es decir que se ejecutan en el navegador del usuario.

[7] **SGML**: consiste en un sistema para la organización y etiquetado de documentos

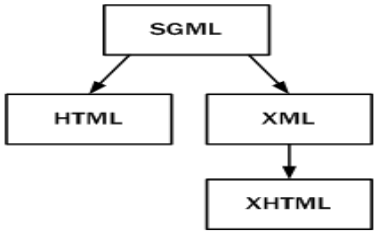

Figura No 2. Esquema de la evolución de HTML y XHTML

En el sistema GINA es utilizada la versión 1.0 del estándar XHTML. Este es el lenguaje básico sobre el cual se encuentra desarrollada la interfaz de usuario del sistema.

CSS:

CSS es un lenguaje de hojas de estilos creado para controlar la presentación de los documentos electrónicos definidos con HTML y XHTML. Es la mejor forma de separar los contenidos y su presentación y es imprescindible para la creación de páginas web complejas. La separación de los contenidos y su presentación posee numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados “documentos semánticos”). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes(30). En el sistema GINA es empleado para decorar el aspecto de la capa de presentación

JavaScript:

Constituye un lenguaje de programación del lado del cliente creado por Brendan Eich en la empresa *Netscape Communications*. Usado en la implementación de páginas web dinámicas, agregándole a estas funcionalidad e interacción. Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. (31)

Entre sus principales características es posible citar:

- Es interpretado por el cliente.
- Está basado en objetos.
- Su código se integra en las páginas XHTML, incluido en las propias páginas.
- Las referencias a objetos se comprueban en tiempo de ejecución, por lo tanto no se compila.

- Es independiente de la plataforma, por lo que se ejecuta en cualquier sistema operativo.

En el sistema GINA se utiliza, principalmente, para manejar objetos dentro de las páginas web. Dichos objetos facilitan la programación de páginas interactivas, a la vez que se evita la posibilidad de ejecutar comandos que puedan ser peligrosos para la máquina del usuario, tales como formateo de unidades y modificación de archivos.

1.8.3.2. Lenguajes utilizados en el lado del servidor

El sistema GINA se ha implementado sobre la base del PHP como lenguaje de programación del lado del servidor[8].

PHP, acrónimo de "*PHP: Hypertext Preprocessor*", es un lenguaje "Open Source" interpretado de alto nivel, especialmente pensado para desarrollos web y el cual puede ser incrustado en páginas HTML. La mayoría de su sintaxis es similar a C, Java y Perl y es fácil de aprender. La meta de este lenguaje es permitir escribir a los creadores de páginas web, páginas dinámicas de una manera rápida y fácil, aunque se pueda hacer mucho más con PHP. (32)

Cuenta con cuatro grandes características que hacen que este lenguaje sea distinguido entre los demás existentes en el mundo:

- Velocidad**:** no sólo la velocidad de ejecución, la cual es importante, sino además no crear demoras en la máquina. Por esta razón no debe requerir demasiados recursos de sistema. PHP se integra muy bien junto a otro software, especialmente bajo ambientes Unix, generalmente es utilizado como módulo de Apache, lo que lo hace extremadamente veloz.
- Estabilidad**:** ninguna aplicación es 100% libre de errores, pero PHP goza de la ayuda de una gran comunidad de desarrollo, permitiendo que los fallos de funcionamiento se encuentren y se reparan rápidamente. PHP utiliza su propio sistema de administración de recursos y dispone de un sofisticado método de manejo de variables, conformando un sistema robusto y estable.
- Seguridad**:** PHP provee diferentes niveles de seguridad, estos pueden ser configurados de archivos de configuración que se encuentran en el servidor.
- Simplicidad**:** es un lenguaje de programación simple de implementar por lo que usuarios con experiencia en C y C++ podrán utilizar PHP rápidamente. (33)

[8] **Lenguajes en el lado del servidor:** son los lenguajes que se procesan en un servidor remoto y que generan la página web antes de enviarla al cliente.

Otra característica que cabe citar además es la conectividad. PHP dispone de una amplia gama de librerías, y agregarle extensiones para extender su alcance es muy fácil. Esto le permite al lenguaje PHP ser utilizado en muchas áreas diferentes, tales como encriptado, gráficos y XML.

Para el desarrollo del sistema GINA se emplea el PHP en su versión 5.2.

1.8.4. Marcos de trabajo (Frameworks)

Se han realizado importantes progresos orientados a la reusabilidad del software a través de la aplicación del paradigma de la programación orientada a objetos y mediante el uso de los componentes tecnológicos. Sin embargo, estas tecnologías sólo proveen el re-uso en el nivel individual, frecuentemente a menor escala. Con la aparición de los patrones de diseño se ha demostrado la reutilización de soluciones para resolver problemas de escala mayor enfocados en la solución de problemas sencillos. Sin embargo, el más complejo problema de la reutilización de soluciones es en el nivel de los grandes componentes, para que estos se puedan adaptar para las solicitudes individuales. (34)

Un *framework*, en el desarrollo de software es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Estos simplifican el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, proporcionan estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener; y facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas.

Framework Ext Js

Ext JS es una librería JavaScript ligera y de alto rendimiento, compatible con la mayoría de navegadores que nos permite crear páginas e interfaces web dinámicas. (35) Permite realizar aplicaciones web enriquecidas basándose en tecnología AJAX, JSON, DHTML y DOM. Con Ext JS, se pueden desarrollar aplicaciones web multiplataforma con facilidad. El modelo de componente de Ext JS mantiene su código bien estructurado por lo que incluso las aplicaciones más grandes pueden ser de fácil mantenimiento. Ext JS brinda la posibilidad de utilizar un gran número de componentes visuales que mejoran considerablemente la calidad de las aplicaciones. Brinda la posibilidad de validaciones de formularios de todo tipo, basándose en expresiones regulares y tipos de datos. Trae implícitos componentes como vista en árboles, arrastrado y soltado, cambio de tamaño de imágenes, rejillas, paginado, agrupado de objetos,

asistentes, entre otros. La utilización de un *framework* de JavaScript como Ext JS facilita la separación de las capas de la vista con la del controlador desde el punto de vista productivo ya que el código utilizado en la primera es solamente JavaScript y no es necesario utilizar ningún tipo de código PHP, así los desarrolladores pueden centrarse más en el aprendizaje de un solo lenguaje. Además, al soportar la serialización de objetos mediante tecnología JSON, permite que los datos enviados desde el controlador como respuesta a la vista contengan sólo las propiedades de dichos objetos, pero no el comportamiento, minimizando los posibles errores de programación y los accidentes de que los objetos sean modificados erróneamente desde la vista. (35)

Para el desarrollo del sistema GINA se utiliza este *framework* en su versión 3.0, principalmente, para validar los datos antes de ser enviados al servidor y para proveer al sistema de una buena apariencia y usabilidad de forma tal que se simule una aplicación de escritorio en el navegador.

Framework Symfony

Symfony es un completo *framework* diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Las clases de la capa del modelo se generan automáticamente, en función de la estructura de datos de la aplicación. La librería Propel[9] se encarga de esta generación automática, ya que crea el esqueleto o estructura básica de las clases y genera automáticamente el código necesario. (36)

La abstracción de la base de datos es completamente transparente para el programador, ya que se realiza de forma nativa mediante PDO (*PHP Data Objects*). Así, si se cambia el sistema gestor de bases de datos en cualquier momento, no se debe reescribir el código, ya que tan solo es necesario modificar un parámetro en un archivo de configuración.

Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Su carácter libre y multiplataforma lo hacen uno de los *framework* de PHP más usados en el mundo. (36)

Symfony se diseñó para que se ajustara a los siguientes requisitos:

[9] **Propel:** es un proyecto de software libre, es una de las mejores capas de abstracción de objetos/relacional disponibles en PHP 5.

- Funciona con todas las bases de datos comunes (MySQL, PostgreSQL, SQLite, Oracle, MS SQL Server).
- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos.
- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador sólo debe configurar aquello que no es convencional.
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web.
- Preparado para aplicaciones empresariales y adaptables a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de *phpDocumentor* y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros. (36)

Este *framework* de desarrollo, en su versión 1.2.8, es la herramienta base utilizada para llevar a cabo la implementación de la lógica de negocio del sistema GINA, proporcionando robustez y seguridad al producto de software.

1.8.5. Entorno de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado (en inglés *Integrated Development Environment* o IDE) es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de estos. (37)

Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes: un editor de texto, un compilador, un intérprete, unas herramientas para la automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones.

NetBeans IDE

NetBeans IDE es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE, además es un producto

libre y gratuito sin restricciones de uso. (38) Posee un amplio soporte para el lenguaje PHP así como para los *framework* de trabajo Symfony y Ext JS.

Para el desarrollo de la solución se ha elegido NetBeans IDE, en su versión 7.0.1, como Entorno de Desarrollo Integrado.

1.8.6. Gestor de Base de Datos

Un Sistema Gestor de Bases de Datos (SGBD) es una colección de programas cuyo objetivo es servir de interfaz entre la base de datos, el usuario y las aplicaciones. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. Un SGBD permite definir los datos a distintos niveles de abstracción y manipular dichos datos, garantizando la seguridad e integridad de los mismos. (39)

En las soluciones implementadas para la AGR el gestor de base de datos que se utiliza es Oracle en su versión 11g. Consiste en un sistema de gestión de base de datos relacional desarrollado por *Oracle Corporation*. Se considera a este producto como uno de los sistemas de bases de datos más completo en el mercado internacional, destacando el soporte de transacciones, la estabilidad, escalabilidad y soporte multiplataforma.

La AGR usa este software desde el año 1997, periodo en el cual ha obtenido experiencias positivas en cuanto al uso de dicho producto.

1.9. Conclusiones parciales

En el presente capítulo han sido abordados los principales conceptos de necesario conocimiento para una correcta comprensión de este trabajo. Se ha realizado un estudio de los principales sistemas existentes para la gestión de acceso de trabajadores externos a sistemas aduaneros, así como las pautas establecidas en el centro de desarrollo CEIGE, lo que ha permitido sentar las bases que fundamentan las tecnologías y herramientas a utilizar en el desarrollo de la presente solución.

Como línea base de la arquitectura se empleó el Modelo de Desarrollo del Departamento de Soluciones para la Aduana como metodología de desarrollo de software, Ext JS en su versión 3.3.0 como marco de trabajo en el lado del cliente, Symfony en su versión 1.2.8 como marco de trabajo en el lado del servidor y como SGBD se empleó Oracle en su versión 11g.

Capítulo 2. Análisis y Diseño de la solución

2.1. Introducción

En el presente capítulo se presenta una propuesta del sistema a desarrollar, se describen las principales características del negocio, el flujo principal de procesos y su modelado. Se realiza además un análisis crítico del flujo de trabajo actual y se exponen los requisitos funcionales definidos para el sistema, enumerando en este sentido las técnicas de captura y validación de requerimientos. Se obtienen además los principales artefactos del diseño que guiarán el posterior proceso de implementación de la solución, haciendo énfasis en el modelo de diseño de la solución.

2.2. Modelado de los procesos del Negocio

A continuación se describen el proceso general de la solución y el subproceso *Adicionar Entidad*, con la finalidad de lograr un mayor entendimiento del negocio que posibilite el desarrollo de un software con la mayor calidad posible.

2.2.1. Descripción del proceso general de la solución

El siguiente diagrama presenta el proceso general de la solución en cuestión, el cual contiene los demás subprocesos que se efectúan para la gestión de acceso de los trabajadores externos a la AGR. Dichos subprocesos son: Gestionar Entidades, Gestionar Relaciones entre Entidades, Gestionar Trabajadores, Enviar Notificación, Reportes de Tripulantes, Reportes de Declarantes, Reportes de Entidades, Gestionar Usuario para Trabajador Externo y Autenticar Usuario.

Desde el subsistema de RC serán iniciados los subprocesos Gestionar Entidades, Gestionar Relaciones entre Entidades, Gestionar Trabajadores, este último desencadena el subproceso Enviar Notificación, el cual concreta una precondición para que pueda realizarse el subproceso Gestionar Usuario para Trabajador Externo en el subsistema de Administración. También pueden ser desencadenados desde este subsistema los subprocesos Reportes de Tripulantes, Reportes de Declarantes y Reportes de Entidades.

En el subsistema de administración se inicializa el proceso Gestionar Usuario para Trabajador Externo como ya se ha mencionado anteriormente, el cual es uno de los procesos críticos del sistema ya que de este depende en gran medida la seguridad y confiabilidad de la aplicación.

Por último, el proceso de Autenticar Usuario, será inicializado por la aplicación Ventanilla Única, y se puede realizar gracias al resultado positivo de varios subprocesos anteriormente mencionados, ya que es uno de los principales fines de la solución.

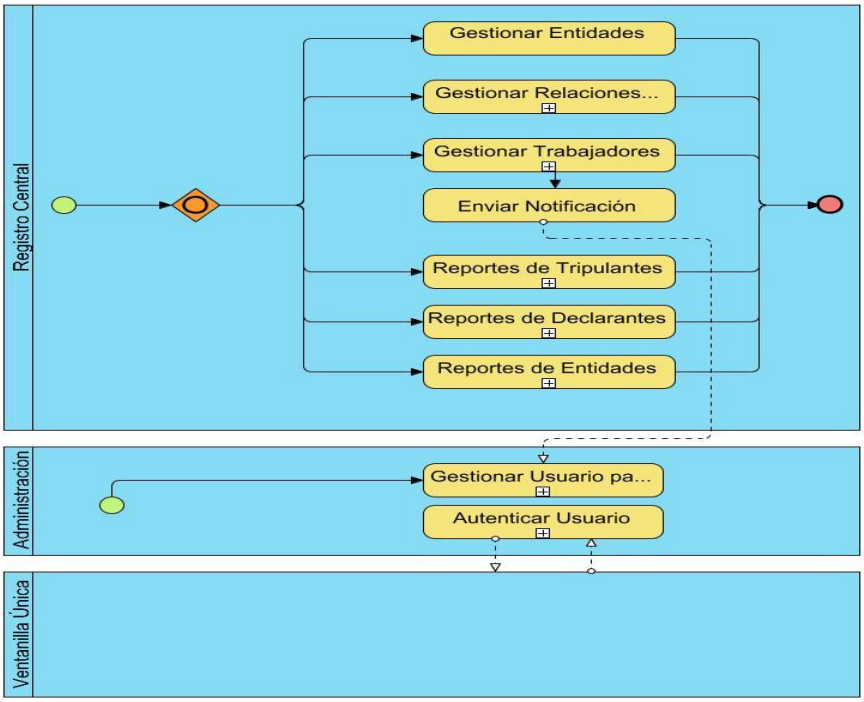

Figura No 3. Modelado del Proceso General de la Solución

2.2.2. Descripción del subproceso Adicionar Entidad

En el siguiente diagrama se muestra el subproceso Adicionar Entidad, parte del subproceso Gestionar Entidad, uno de los subproceso críticos de la solución, pues es uno de los pilares de la misma. El subproceso será iniciado por el usuario que desea realizar la acción, este inserta los datos necesarios y el sistema a continuación hace las verificaciones pertinentes de los mismos, en caso de no existir problemas son almacenados los contactos, el representante y los demás datos de la entidad; en caso de existir un error durante el proceso se muestra una notificación del mismo al usuario.

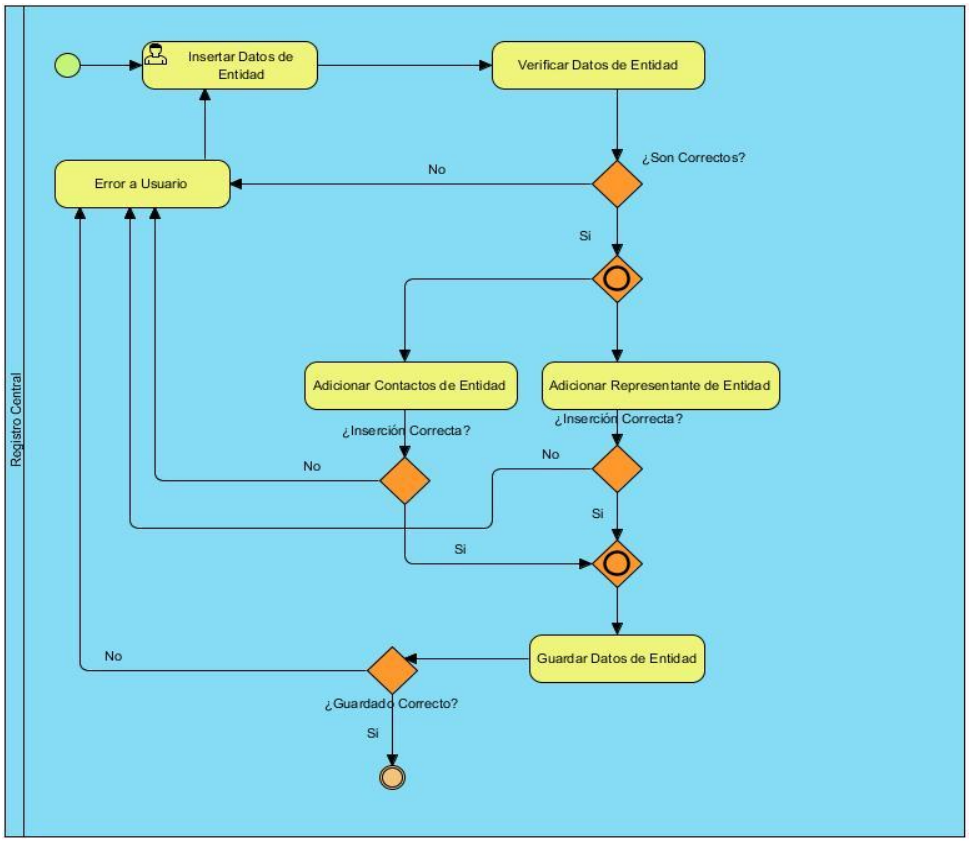

Figura No 4. Modelado del Subproceso Adicionar Entidad

2.3. Análisis Crítico de la Ejecución de los Procesos

Para el correcto flujo de trabajo aduanero es necesaria la interacción con la AGR de entidades que de una u otra forma proveen o necesitan servicios de la misma, y estas lo hacen mediante representantes o trabajadores. Actualmente dichas entidades y/o trabajadores son gestionados de forma poco eficaz en el RCE de la AGR, dado que las herramientas y métodos usados no son capaces de manejar eficiente y correctamente los grandes volúmenes de información existentes. Junto a esto existe un problema aún mayor y es el referente a la gestión de acceso a la información; puesto que se requiere el acceso a la información en diferentes niveles, lo que actualmente no se garantiza, afectando así diferentes partes del proceso. Si se desarrollara una solución que cumpliera con todas las necesidades del proceso, se lograría mayor eficiencia y rapidez en el mismo y un proceso más seguro y confiable.

2.4. Modelo Conceptual

El diseño de una base de datos es un proceso complejo que abarca decisiones a varios niveles. La complejidad se controla mejor descomponiendo el problema general en sub problemas y resolviendo estos sub problemas de forma independiente y usando métodos específicos para cada caso. Así, el diseño es descompuesto en diseño conceptual, lógico y físico.

El diseño conceptual parte de las especificaciones de requisitos de usuario y su resultado es el esquema conceptual de la base de datos. Un esquema conceptual es una descripción de alto nivel de la estructura de la base de datos, independientemente del SGBD que se vaya a utilizar para su manipulación. Un modelo conceptual es un lenguaje que se utiliza para describir esquemas conceptuales. El objetivo del diseño conceptual es describir el contenido de la información de la base de datos y no las estructuras de almacenamiento que se necesitarán para manejar dicha información. El modelo conceptual de la presente solución, puede observarse en el **anexo # 1**.

2.5. Especificación de los Requerimientos del Sistema

Especificar los requerimientos del sistema sirve como base para las actividades de ingeniería de software subsecuentes, describe la función y el desempeño de un sistema basado en computadora y las restricciones que regirán su desarrollo, por lo cual deben tenerse en cuenta las técnicas que son más factibles utilizar para la captura y la validación de estos requisitos; a continuación se exponen las técnicas empleadas además de un listado de los mismos.

2.5.1. Técnicas para la Captura de Requerimientos

Siguiendo el procedimiento para la Ingeniería de Requisitos en el Departamento de Desarrollo de Soluciones para la Aduana del CEIGE, el cual plantea sobre las técnicas para la captura de requerimientos lo siguiente:

Teniendo en cuenta las características del cliente, la AGR, se decidió utilizar las siguientes técnicas que permitirán a los analistas, la recopilación y obtención de la información necesaria para la captura de requisitos, las mismas son: entrevista, observación, tormenta de ideas y talleres. (22)

Se combinan las siguientes técnicas: entrevista, observación y tormenta de ideas. Se llevó a cabo un proceso de **observación** para percibir como se desarrollan las operaciones en el RCE, pudiendo captar directamente las particularidades de estos procesos en las visitas realizadas. Se **entrevistaron**

trabajadores del RCE y funcionarios del Departamento de Técnicas Aduaneras de la AGR, ya que las entrevistas permiten un intercambio más abierto con los especialistas, mediante preguntas que esclarecen con precisión el funcionamiento de todo el negocio; y la **tormenta de ideas** se llevó a cabo en conjunto con especialistas de la AGR acumulando ideas para tener una perspectiva general de las necesidades del sistema.

2.5.2. Requerimientos Funcionales

Los requerimientos funcionales (RF) son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer. (22)

Son capacidades o condiciones que el sistema debe cumplir, describen con detalle la función de éste, sus entradas y salidas, excepciones, entre otros, estos deben estar bien determinados y ser convenientemente comprendidos tanto por los implicados para con el sistema como por los desarrolladores del mismo para que exista un entendimiento común.

A continuación se exponen los requerimientos definidos para el sistema modelado:

- Gestionar Entidad.
  - Adicionar Entidad.
  - Cancelar Entidad.
  - Modificar Entidad.
  - Suspender Entidad.
  - Activar Entidad.
  - Mostrar Entidad.
- Gestionar Relaciones entre entidades (las relaciones entre entidades pueden ser “Prestación Apoderado”, “Asociación Económica”, “Beneficiario Importación”).
- Gestionar Trabajador Externo.
  - Adicionar (cuando se adiciona se le asigna su estado y su tipo, además de los otros elementos de los trabajadores externos, como los contactos).
  - Modificar Trabajador Externo.
  - Mostrar Trabajador Externo.

- Activar Trabajador Externo.
- Suspender Trabajador Externo.
- Cancelar Trabajador Externo.

➢ Enviar Notificación (Cuando se hace una modificación sobre un trabajador).

➢ Gestionar usuario para trabajador externo.

- Adicionar usuario de trabajador externo.
- Modificar usuario de Trabajador Externo.
- Mostrar usuario de Trabajador Externo.
- Cancelar usuario de Trabajador Externo.
- Asignar rol a usuario de trabajador externo.
- Gestionar permisos de un rol de trabajador externo.

➢ Reportes Tripulantes.

- Parte general de tripulantes.
- Total Tripulantes Activos (general y por entidades, sólo devuelve número).
- Listado de tripulantes Activos.
- Listado de tripulantes Bajas.
- Listado de tripulantes no activos.
- Listado de tripulantes con vehículos.

➢ Reportes Declarantes

- Histórico declarante.
- Entidades que ha declarado.
- Fechas de recalificación.

➢ Reportes Entidad.

- Histórico entidad.
- Declarantes según tipo.
- Declarantes suspendidos por ilegalidades.
- Declarantes según tipo.
- Cantidad de apoderados de una empresa.
- Empresas Por Actividad autorizada (Impo / Expo).
- Empresas por Ministerio.

- Empresas por Tipo (Privadas, Mixtas y Estatales).
- Asociaciones Económicas Internacionales.
- Agencias de Aduanas (con sus Agentes).
- Agentes de Aduanas por Agencias.
- Sucursales Sociedades Mercantiles Extranjeras.
- Representaciones Bancarias.
- Agencias de Viajes.
- Agencias Navieras / Agencias Transitarias.
- Agentes de Sociedades Mercantiles Extranjeras.
- Entidades actualizadas según fecha.
- Entidades desactualizadas según fecha.
- Declarantes actualizados según fecha.
- Declarantes desactualizados según fecha.

2.5.3. Técnicas para la Validación de Requerimientos

Siguiendo el procedimiento para la Ingeniería de Requisitos en el Departamento de Desarrollo de Soluciones para la Aduana del CEIGE, el cual plantea sobre las técnicas para la validación de los requerimientos lo siguiente: “En el procedimiento propuesto se recomienda el uso de varias técnicas para la correcta validación de los requisitos en el Departamento de Soluciones para la Aduana las cuales son: revisiones del documento de requerimientos, construcción de prototipos, matrices de trazabilidad y generación de casos de pruebas” (40), la validación de los requerimientos se realizó con la combinación las siguientes técnicas: revisiones del documento de requerimientos y construcción de prototipos. Se realizaron diversas **revisiones al documento de requerimientos** con la participación de los analistas del proyecto, jefe del subsistema, jefe del proyecto y especialistas del CADI para logar una correcta interpretación de la información transmitida, los señalamientos planteados fueron recogidos y aplicados posteriormente. Además se efectuó la **construcción de prototipos**, los cuales fueron presentados por cada requerimiento de software, aclarando con la Especialista Principal de Sistemas Automatizados del CADI, si las necesidades del área de trabajo Depósitos de Aduana fueron cubiertas por el sistema.

2.6. Aportes de la Solución y Beneficios Esperados

La solución propuesta garantiza el cumplimiento de todas las funcionalidades necesarias para una correcta ejecución del flujo de trabajo en el RCE de la AGR, teniendo como premisa la seguridad, eficiencia y claridad en el proceso. Proveerá reportes que garanticen la toma de decisiones oportuna y rapidez en el procesamiento de informes, los cuales servirán para un mejor análisis de las diferentes situaciones que pueden surgir durante todo proceso y evitar además la comisión de delitos.

2.7. Diseño del Sistema

La etapa de diseño es la principal encargada de la futura calidad de la solución, en esta etapa se fomentará la calidad en la ingeniería del software, se proporcionan las representaciones del software susceptibles de evaluar respecto de la calidad y es la única forma en que, de manera exacta, un requisito del cliente se puede convertir en un sistema o producto de software terminado.

El diseño del software sirve como fundamento para todas las actividades subsecuentes de la ingeniería de software y del soporte de este. Sin una buena etapa de diseño se corre el riesgo de construir un sistema inestable, el cual fallará cuando se realicen cambios pequeños; que será difícil de probar; cuya calidad no podrá evaluarse sino hasta etapas tardías del proceso del software.

A continuación se describen los objetivos principales de dicha etapa:

- Crear una entrada apropiada y un punto de partida para actividades de implementación subsiguientes capturando los requisitos o subsistemas individuales, interfaces y clases.
- Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo, teniendo en cuenta la posible concurrencia.
- Adquirir una comprensión en profundidad de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia, tecnologías de interfaz de usuario, tecnologías de gestión de transacciones, etc.
- Visualizar y reflexionar sobre el diseño utilizando una notación común. (41)

2.7.1. Patrones utilizados

Con la utilización del marco de trabajo Symfony, el sistema a desarrollar implementa varios de los patrones de diseño y arquitectónicos utilizados en la actualidad, ofreciendo a los desarrolladores la

utilización de buenas prácticas en la implementación y la no preocupación de la aplicación de los mismos, brindándole al equipo de desarrollo y al producto cuantiosas ventajas.

Patrones GRASP

- Controlador**:** Symfony implementa un controlador frontal para atender todas las peticiones web (*sfActions*), es el único punto de entrada de toda la aplicación en un determinado entorno. Cuando recibe una petición, utiliza el sistema de enrutamiento para enviar la acción al controlador responsable de la solución y se encarga de manejar la seguridad y ejecución de los filtros. Este patrón se evidencia en las clases *sfFrontController*, *sfWebFrontController*, *sfContex*, los "*actions*" y el *index.php del ambiente.*
- Alta Cohesión**:** Symfony permite la organización del trabajo en cuanto a la estructura del proyecto y la asignación de responsabilidades con una alta cohesión. Un ejemplo de ello es la clase *aduanaActions*, es responsable de definir las acciones para las plantillas y colaborar con otras para realizar diferentes operaciones, instanciar objetos y acceder a las propiedades. Está formada por varias funcionalidades que están estrechamente relacionadas.
- Bajo Acoplamiento**:** la clase *aduanaActions* hereda únicamente de *sfActions* para alcanzar un bajo acoplamiento de clases. Las clases que implementan la lógica del negocio y de acceso a datos se encuentran en el modelo, las cuales no tienen asociaciones con las de la vista o el controlador, lo que proporciona que la dependencia en este caso sea baja.
- Experto**:** Symfony utiliza Propel para realizar su capa de abstracción en el modelo, encapsular toda la lógica de los datos y generar las clases con todas las funcionalidades comunes de las entidades, las clases de abstracción de datos (Peer del Modelo) poseen un grupo de funcionalidades que están relacionadas directamente con la entidad que representan y contienen la información necesaria de la tabla que representan.
- Creador**:** en la clase *aduanaActions* se localizan las acciones definidas para el módulo, en las cuales se crean los objetos de las clases que representan las entidades, confirmando que la clase es "creador" de dichas entidades.

Patrones GoF

- Instancia Única**:** la clase *sfRouting* presenta el método *getInstance()*, es utilizada por el controlador frontal (*sfWebFrontController*) y se encarga de enrutar todas las peticiones que se hagan a la

aplicación. *sfRouting* garantiza la existencia de una única instancia y la creación de un mecanismo de acceso global a dicha instancia.

- Comando**:** se encuentra presente en la clase *sfWebFrontController*, en el método *dispatch()*,es la encargada de establecer el módulo y la acción que se va a usar según la petición del usuario. Además se aplica en la clase *sfRouting*, que está desactivada por defecto y procede según las necesidades del administrador del sistema donde se aplique el framework, la cual se puede activar o desactivar.
- Decorador: la clase abstracta *sfView*, padre de todas las vistas, contienen un decorador para permitir agregar funcionalidades dinámicamente. El archivo *layout.php*, conocido como plantilla global, contiene el código HTML que es tradicional en todas las páginas del sistema, para no tener que repetirlo. El contenido de la plantilla se integra en el *layout*, o si se mira desde el otro punto de vista, el *layout* decora la plantilla.

2.7.2. Patrón MVC según Symfony

El patrón MVC obliga a dividir y organizar el código de acuerdo a las convenciones establecidas por el framework, todo lo relacionado con la interfaz de usuario se guarda en la vista, la manipulación de datos se guarda en el modelo y el procesamiento de las peticiones constituye el controlador. El empleo del patrón MVC en un sistema resulta bastante útil además de restrictivo.

Symfony está basado en el patrón de arquitectura conocido MVC, formado por tres niveles: el *Modelo* representa la información con la que trabaja la aplicación, es decir, su lógica de negocio, la *Vista* es la encargada de originar las páginas que son mostradas como resultado de las acciones y el *Controlador* se encarga de procesar las interacciones del usuario y realiza los cambios apropiados en el modelo o en la vista. (36)

La implementación que realiza Symfony de la arquitectura MVC incluye varias clases como son:

- sfController: es la clase del controlador y se encarga de decodificar la petición y transferirla a la acción correspondiente.
- sfRequest: guarda todos los elementos que integran la petición (parámetros, cookies, cabeceras entre otros).
- sfResponse: posee las cabeceras de la respuesta y los contenidos. El contenido de este objeto se convierte en la respuesta HTML que se remite al usuario.

- El singleton de contexto (que se obtiene mediante ***sfContext****::getInstance()*): guarda una referencia a todos los objetos que constituyen el núcleo de Symfony y puede ser accedido desde cualquier parte de la aplicación.

2.7.3. Modelo del Diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de usos centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen un impacto en el sistema a considerar. Además, el modelo de diseño sirve como abstracción de la implementación del sistema y es, de ese modo, utilizado como una entrada fundamental de las actividades de implementación. (31)

El modelo del diseño de la presente investigación se encuentra compuesto por el diagrama de paquetes, el modelo de datos y por los diagramas de secuencia orientados a las actividades.

2.7.3.1. Diagrama de Paquetes

Un diagrama de paquetes es un conjunto de agrupaciones lógicas de elementos del modelo, mostrando las dependencias entre dichas agrupaciones. Los paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Un paquete puede contener paquetes subordinados, así como otros tipos de elementos del modelo. Todo tipo de elementos del modelo UML pueden ser organizados en paquetes.

En la siguiente figura se muestra el diagrama de paquetes de la presente solución, este está compuesto por los paquetes que serán extendidos en búsqueda de los resultados esperados, estos son el plugin de administración suAdminPlugin y el subsistema de RC; además está compuesto por el subsistema Tablas de Control, el cual gestiona todos los nomencladores genéricos del sistema GINA, y en general todos estos subsistemas hacen uso de las librerías del framework Symfony.

Es válido destacar además un paquete que interviene en la solución y es el caso de la Ventanilla Única, el cual hará uso de varios servicios brindados por la solución y será uno de los principales beneficiados de la misma y a quien mayoritariamente estará orientada.

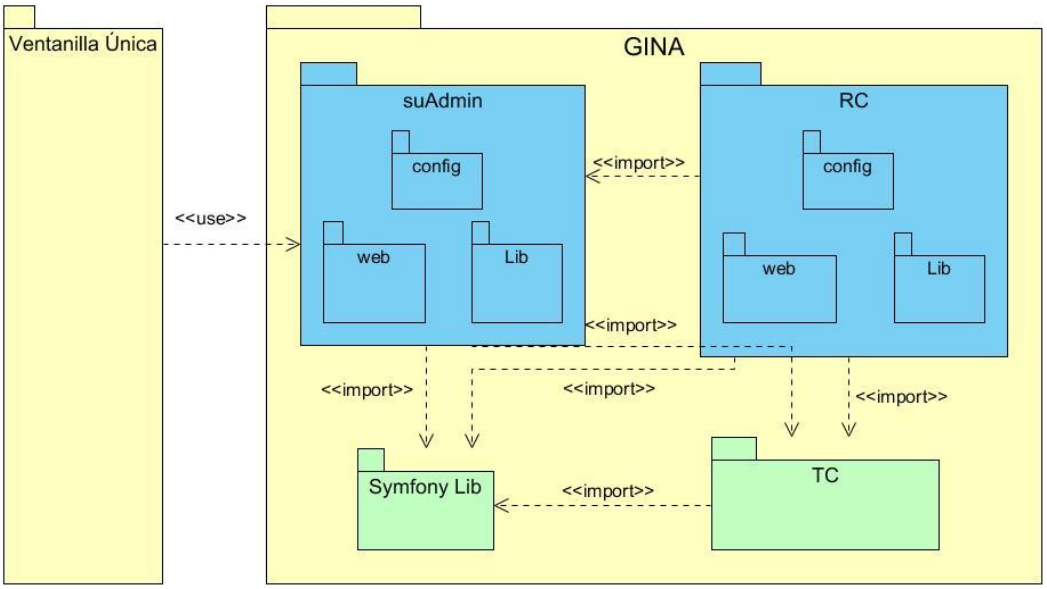

Figura No 6. Diagrama de Paquetes

2.7.4. Diseño de la Base de Datos

En términos generales el objetivo del diseño de una base de datos relacional es generar los esquemas de relaciones que permitan el almacenamiento de la información con un mínimo de redundancia de la misma y a su vez facilitar su recuperación.

El modelo Entidad-Relación (ER) fue propuesto por Peter P. Chen en los años 1976 y 1977, y posteriormente el tema ha sido estudiado por varios autores que han hecho excelentes aportes, por lo que se puede asegurar que no existe un único modelo Entidad-Relación.

Este modelo describe los datos como entidades, relaciones (vínculos) y atributos, y permite representar el esquema conceptual de una base de datos de forma gráfica mediante los diagramas ER.

En el anexo # 2, se puede observar el diagrama ER de la presente solución, el cual en su totalidad está compuesto por 41 tablas, divididas en diferentes colores para mejor entendimiento del mismo. Las tablas de color gris son tablas de nomencladores ubicadas en el esquema del subsistema de Tablas de Control; las tablas de color verde son las ubicadas en el esquema del subsistema de Administración y las de color naranja las ubicadas en el subsistema de RC.

2.7.5. Diagramas de Secuencia Orientados a las Actividades.

El diagrama de secuencia orientado a actividades, tiene como principal meta definir con mayor exactitud

la relación existente entre las funcionalidades que se deben implementar y las clases correspondientes donde deben ser hechas. Este se nutre de las principales bondades que brindan el diagrama de Actividades y el Diagrama de Secuencia. Uno de los objetivos que se persiguen con dicho diagrama es facilitar el trabajo de los programadores que no tienen un alto nivel de involucración en el desarrollo del proyecto, pudiéndose así comprender con facilidad el diseño de cada una de las actividades orientadas por los diseñadores a cualquiera de los programadores del departamento.

Entre sus principales características se puede identificar cómo declarar funcionalidades y variables, así como la especificación de los parámetros establecidos para las funciones, que a su vez muestran su visibilidad y estructura interna.

En el anexo # 3, es posible observar una muestra de este diagrama.

2.8. Conclusiones parciales.

Durante el presente capítulo se hizo la modelación y descripción de los procesos y subprocesos del negocio con el objetivo de lograr un correcto entendimiento de los mismos, convirtiéndose dichos artefactos en la base de la fase de diseño, en la cual se generó otro grupo de artefactos que definen la estructura que tendrá la solución y que a su vez consisten en la base para el posterior proceso de implementación de la solución, la cual para una correcta y fructífera ejecución requiere precisamente de cimientos sólidos.

Capítulo 3. Implementación y Pruebas de la SoluciónIntroducción

En el presente capítulo se abordará todo lo referente a la implementación de la solución, por lo cual se evidenciará la utilización de patrones de diseño en el código fuente del mismo. Esto se logrará mediante la traducción del diseño en la solución y para ello se hará uso de los diagramas de interacción como base para la codificación. En el proceso de desarrollo de esta etapa se garantizará la calidad de la solución mediante la aplicación de pruebas de software, lo cual permitirá revelar la calidad del producto dado diferentes indicadores como la usabilidad y calidad.

3.2. Estándares de Codificación

Los estándares de codificación no son más que una guía que define la estructura y composición sobre la cual se debe regir la codificación de una solución informática. Su objetivo fundamental es lograr un correcto, común y comprensible formato del código que compone un sistema. Este estándar debe servir de apoyo a quien interactúa con dicho código para identificar de forma sencilla cuál es el objetivo y las funcionalidades que brinda cada una de las clases, funciones y demás componentes de software, además que debe servir de guía para posteriores implementaciones y/o modificaciones del sistema.

Para definir el estilo de codificación a seguir en la solución se utilizó la notación estándar establecida para aplicaciones desarrolladas con PHP (PHP Coding Standard), que mayormente está basada en el estándar de código de aplicaciones desarrolladas en C++ (C++ Coding Standard). A continuación se muestran las principales reglas de codificación utilizadas en la implementación de la solución, teniendo como base además el documento Propuesta de un Estándar de Codificación del Departamento de Soluciones para la Aduana del CEIGE.

3.2.1. Reglas Generales de Codificación

La apertura y cierre de las etiquetas de PHP se harán de la siguiente forma:

```
<?php
 //código PHP
?>
```

Los encabezados de las funciones y acciones deben tener las especificaciones principales de la misma, dígase requisito al que da solución, los parámetros de entrada, la respuesta, así como otros elementos que se consideren importantes:

```
/**
```

```
*RF: Insertar Usuario
*@param Array $paramsUsuario
*Arreglo de parámetros con los datos del nuevo usuario
*@return boolean:
*true o false en dependencia del resultado de la acción realizada
**\
```

Los nombres de las funciones, variables y demás elementos de codificación deberán estar en nomenclatura CamelCase[10] , además de especificar de forma rápida, sencilla y con la menor cantidad de palabras su objetivo o función:

```
//arreglo de usuarios
$arregloUsuarios;
//función para obtener los permisos de un rol
static function obtenerPermisosRol($rol){
}
```

En el caso de las acciones del marco de trabajo, y siguiendo las especificaciones del mismo, estas deben iniciar con el prefijo *execute*, por ejemplo:

```
executeInsertarUsuario(), ó executeEliminarEntidad.();
```

3.3. Tratamiento de Errores

Para garantizar el correcto funcionamiento de cualquier sistema es imprescindible identificar y controlar los posibles errores que se pueden presentar durante la ejecución de las diferentes tareas del software y durante su interacción con el usuario. En la implementación de la solución se tratan los errores de forma tal que las interacciones con la base de datos se hacen de forma segura y correcta, no alterando así su correcto estado, además de controlar los diferentes errores que puedan surgir en el sistema en tiempo de ejecución.

Lo anteriormente comentado se logra mediante los diferentes mecanismos de validación que comprueban la autenticidad y estado correcto de los datos que maneja el sistema tanto en el lado del cliente como en el lado del servidor.

En el lado del cliente se diseñaron las interfaces de forma tal que el usuario introduzca la menor cantidad de datos posibles, evitando así posibles incoherencias y errores en los mismos; además fueron

[10] CamelCase: estilo de escritura que se aplica a frases o palabras compuestas.

implementadas funciones que validan dichos datos y de existir errores muestran mensajes al usuario alertando los errores detectados.

En el lado del servidor se hizo uso de los validadores de formularios introducidos en el framework Symfony desde su versión 1.2. Estos permiten una estrecha relación entre los atributos de la base de datos y los que se necesiten introducir, haciendo uso de los mecanismos de excepciones del lenguaje PHP en caso de detección de errores.

Fueron utilizadas además las instrucciones *try catch,* para el manejo de excepciones y errores en otros momentos de la ejecución de tareas.

3.4. Diagramas de Despliegue

El diagrama de despliegue se utiliza para modelar la configuración de los elementos de procesado en tiempo de ejecución y de los componentes, procesos y objetos de software que viven en ellos. Se modelan los nodos físicos y las asociaciones de comunicación que existe entre ellos. (42)

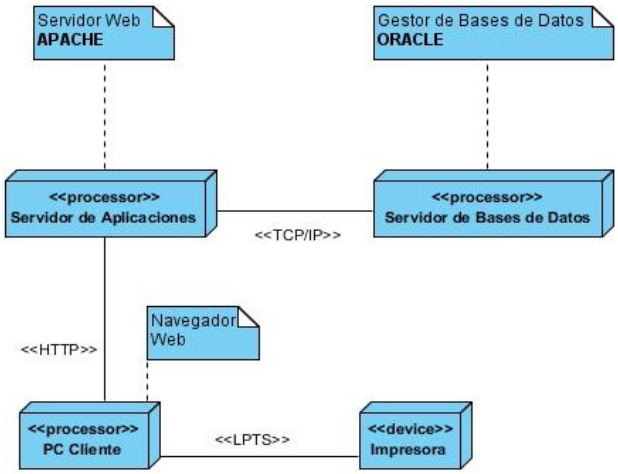

Figura No 9. Diagrama de Despliegue de la Solución

Nodo PC Cliente

Representa las computadoras que utilizarán los usuarios para interactuar con la aplicación. Establece comunicación con el servidor de aplicaciones a través del protocolo HTTP.

Nodo Impresora

Representa las impresoras, pues en algunos casos se imprimen reportes generados por el sistema.

Nodo Servidor de Aplicaciones

En este nodo se encuentran los scripts de la aplicación.

Nodo Servidor de Base Datos

En este nodo se encuentra el Servidor de Base de datos del sistema GINA con todos sus subsistemas.

3.5. Diagramas de Componentes

Los componentes constituyen la parte modular del sistema, encapsulan implementación y un conjunto de interfaces y proporciona la realización de los mismos. Un componente típicamente contiene clases y puede ser implementado por uno o más artefactos (ficheros, ejecutables, binarios, tablas y documentos). Constituyen recursos desarrollados para un fin concreto y que puede formar a parte o junto con otros, un entorno funcional requerido por cualquier proceso predefinido. Son independientes entre ellos, y tienen su propia estructura e implementación.

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre los mismos. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Los diagramas de componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema.

En el anexo # 4, se muestra el diagrama de paquetes de la solución, el cual está compuesto por dos subsistemas, Administración y RC, estos compuestos por paquetes que aunque su denominación y función dentro de los subsistemas sea la misma, tienen diferentes implementaciones en dependencia de las funcionalidades que se persiguen en cada subsistema; además ambos usan de forma común los paquetes de librerías de interfaces gráficas así como el núcleo de Symfony.

Los paquetes “Mapeo de Datos”, poseen la abstracción del modelo de datos de cada subsistema, los paquetes “Configuración” contienen los ficheros de configuración genéricos de cada subsistema y los “Interfaces” los ficheros de interfaces gráficas. Por último, ambos subsistemas poseen las clases controladoras actionClass.php, las cuales son precisamente el centro de control de todos los procesos.

3.6. Uso de los diferentes patrones durante la implementación

Como ya se mencionó en el capítulo 1, epígrafe 1.7.4, existen en el mundo del desarrollo de software un grupo de patrones, principios generales o estándares que proporcionan métodos muy efectivos en la producción de software, en el presente epígrafe se hace una pequeña explicación del uso de algunos de estos durante el desarrollo de la solución, en el caso del patrón de diseño MVC, se abordará en el siguiente epígrafe (3.7) de forma un poco más práctica para mejor entendimiento.

Adapter

Problema: ¿Cómo tener una única interfaz de acceso a múltiples bases de datos?

Propósito: Convertir la interfaz de una clase para que se adapte a lo que el cliente que la usa necesita, permitiendo así que trabajen juntas clases cuyas interfaces son incompatibles.

Solución: Symfony da la posibilidad de cambiar a otro sistema de base de datos completamente diferente a mitad de desarrollo, sólo basta con configurar un archivo YAML. La capa de abstracción utilizada encapsula toda la lógica de los datos. El resto de la aplicación no tiene que preocuparse por las consultas SQL y el código SQL que se encarga del acceso a la base de datos.

Singleton

Problema: Obtener un punto de acceso global a una clase.

Propósito: Asegurar que sólo exista una instancia de una clase especifica en un sistema a desarrollar.

Solución: El controlador frontal proporciona un punto de acceso global a la aplicación. En una acción, el método *getContext()* devuelve el mismo *singleton*. Se trata de un objeto muy útil que guarda una referencia a todos los objetos del núcleo de Symfony relacionados con una petición dada, y ofrece un método de acceso para cada uno de ellos. Algunos ejemplos utilizados en la aplicación.

Propel::getConnection('tc');

SysComponentsLocator::getInstance();

Bajo acoplamiento

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?

Propósito: Aumentar la reutilización y eliminar las dependencias entre las clases para propiciar un fácil mantenimiento y entendimiento.

Solución: Symfony asigna a cada clase una responsabilidad para mantener pocas dependencias entre las mismas. La aplicación desarrollada mantiene el sistema de clases generadas por symfony y sus dependencias.

Alta cohesión

Problema: ¿Cómo mantener la complejidad dentro de límites manejables?

Propósito: Cada elemento dentro del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable.

Solución: Symfony agrupa las clases por funcionalidades que son fácilmente reutilizables, bien por su uso directo o por herencia.

Controlador

Problema: ¿Quién debería encargarse de un evento del sistema?

Propósito: Facilitar la centralización de actividades, delegar las actividades en otras clases con las que mantiene un modelo de alta cohesión.

Solución: Symfony implementa para cada vista de la aplicación una clase controladora que se encarga de atender el evento generado por la vista. Lo cual se evidencia en la clase *sfActions* del módulo Registro Central.

Controlador Frontal

Problema: ¿Quién debería encargarse de manejar las peticiones de los usuarios?

Propósito: Centralizar todas las peticiones de los usuarios y proveerlos de un punto único de entrada a la aplicación.

Solución: Todas las peticiones web son manejadas por un sólo controlador frontal, que es el punto de entrada único de toda la aplicación en un entorno determinado. Cuando el controlador frontal recibe una petición, utiliza el sistema de enrutamiento para enviar la acción al controlador responsable de la solución. Además de esto el controlador frontal se encarga de manejar la seguridad y la ejecución de los filtros en Symfony.

3.7. Comunicación entre capas

Como anteriormente se especifica, para el desarrollo de la solución se hace uso del patrón de arquitectura Modelo-Vista-Controlador (MVC), y específicamente el framework Symfony lo utiliza logrando mejoras significativas en rendimiento, escalabilidad y organización de las diferentes estructuras del sistema. A continuación se muestra un ejemplo de comunicación entre capas para ayudar a comprender el funcionamiento de la solución.

3.7.1. Ejemplo de Comunicación entre Capas

En el presente epígrafe se explica el funcionamiento de la comunicación entre capas del sistema durante la ejecución del proceso de gestión de notificaciones de usuarios.

Este se inicia cuando un usuario se autorizado llega a la interfaz de gestión de usuarios externos:

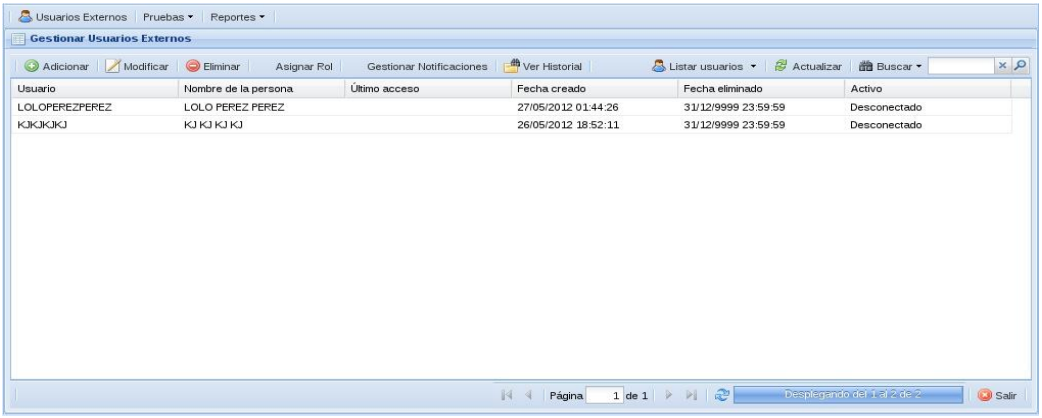

Figura No 11. Interfaz de Muestra de Usuarios Externos

En este momento se dispara una petición automática al servidor para conocer si existen notificaciones pendientes a procesar, a continuación se muestra un fragmento de dicho código, en el cual se hace la petición a la acción *executeJsonGridNotificaciones()* en la clase controladora:

```
Petición Ajax hecha desde el cliente a la clase controladora
Ext.Ajax.request({
        url: 'externo/jsonGridNotificaciones',
        success:function(response, request){
            var jsonData = Ext.util.JSON.decode(response.responseText);
            if(jsonData.cantidad !=0){
                Ext.MessageBox.confirm('Atención', 'Tiene nuevas notificaciones ¿Desea atenderlas?',      function
AtenderNotificaciones(btn){
                    if (btn == 'yes') {
                        new NotificacionesArc().show();
                    }
                });

            }
        },
        failure:function(){
            Ext.MessageBox.alert('Error', 'Falló la conección con el servidor, por favor revise su conección de red')
        }

    }); )
```

En la clase controladora *actions.class.php*, es recibida esta petición y se procesa, instanciando en su cuerpo la funcionalidad *obtenerNotificaciones()* de la clase del modelo *ArcNotifTrabExtPeer.php*:

```
/*    * (PHP5)
     * Función que devuelve un json con las notificaciones pendientes
     * @return Json
     * Json con los datos de las notificaciones pendientes
     * @autor Leodan Rodriguez Diaz lrodriguezd@estudiantes.uci.cu
     */

    public function executeJsonGridNotificaciones() {
        $notificaciones = ArcNotifTrabExtPeer::obtenerNotificaciones();
        $msg = $notificaciones['msg'];
        $json = $notificaciones['notificaciones'];
        $cantidad = $notificaciones['cantidad'];
        return $this->renderText("{cantidad: '" . $cantidad . "', msg: '" . $msg . "', notificaciones: " . json_encode($json) . "}");
    }
```

En la clase *ArcNotifTrabExtPeer.php* se ejecuta la función instanciada:

```
static public function obtenerNotificaciones() {
        $criteria = new Criteria();
        $criteria->addAscendingOrderByColumn(self::ID_NOTIF_TRAB_EXT);
        $total = self::doCount($criteria);
        $notificaciones = self::doSelect($criteria);
        $resultado['notificaciones'] = array();
        $resultado['msg'] = '';
        if ($total > 0) {
            $syscomponent = SysComponentsLocator::getInstance();
            $componenteTc = $syscomponent->getComponent('tc');
            $componentePersona = $syscomponent->getComponent('persona');
            foreach ($notificaciones as $notificacion) {
                $arrayNotificacion = $notificacion->toArray();
                //resuelvo tipo trab
                $criteria = new Criteria();
                $criteria->add(TcTipoTrabExtPeer::ID_TIPO_TRAB_EXT, $arrayNotificacion['TipoTrabajador']);
                $paramsTipoTrabajador = array('estado' => SisTcTablaPeer::VIGENTE, 'nombreTc' =>
TcTipoTrabExtPeer::TABLE_NAME, 'criteria' => $criteria, 'encode' => false);
                $tipoTrabajador = $componenteTc->executeService(new sfEvent(null, 'tc.obtenerDadoCriteria',
$paramsTipoTrabajador));
                //resuelvo la operacion
//                $criteria = new Criteria();
//                $criteria->add(TcTipoOperacionArcPeer::ID_TIPO_OPERACION_ARC,
$arrayNotificacion['IdTipoOperacionArc']);
//                $paramsTipoOperacion = array('estado' => SisTcTablaPeer::VIGENTE, 'nombreTc' =>
TcTipoOperacionArcPeer::TABLE_NAME, 'criteria' => $criteria, 'encode' => false);
```

```
//              $tipoOperacion = $componenteTc->executeService(new sfEvent(null, 'tc.obtenerDadoCriteria',
$paramsTipoOperacion));
            $criteria = new Criteria();
            $criteria->add(TcTipoOperacionArcPeer::ID_TIPO_OPERACION_ARC,
$arrayNotificacion['IdTipoOperacionArc']);
            $tipoOperacion =TcTipoOperacionArcPeer::doSelect($criteria);
            if($tipoOperacion){
               $arrayNotificacion["codigoOperacionArc"] = $tipoOperacion[0]->getCodigo();
            }else{
               throw new Exception("Error al obtener el tipo de operaci&oacute;n a realizar");
            }


            $persona = $componentePersona->executeService(new sfEvent(null, 'persona.buscarPorIdPersona',
array("idPersona" => $arrayNotificacion["IdPersona"])));
            $docIdentif = $componentePersona->executeService(new sfEvent(null, 'persona.obtenerDocIdentifPorId',
array("idPersona" => $arrayNotificacion["IdPersona"])));
            if ($docIdentif)
              $arrayNotificacion["CI"] = $docIdentif[0]->getNumero();
            $arrayNotificacion["CodigoTipoTrabajador"] = $tipoTrabajador[0]->getCodigo();
            $arrayNotificacion["PrimerNombre"] = $persona->getPrimerNombre();
            $arrayNotificacion["SegundoNombre"] = $persona->getSegundoNombre();
            $arrayNotificacion["PrimerApellido"] = $persona->getPrimerApellido();
            $arrayNotificacion["SegundoApellido"] = $persona->getSegundoApellido();
            //$arrayNotificacion["CodigoTrabajadorExterno"] = "TE_4fc06202b7340";
            array_push($resultado ['notificaciones'], $arrayNotificacion);
         }
         $msg = "Usted tiene " . $total;
         if ($total == 1) {
            $msg .= " solicitud pendiente";
         }
         if ($total > 1) {
            $msg .= " solicitudes pendientes";
         }
         $resultado ['msg'] = $msg;
      }
      $resultado ['cantidad'] = $total;
      return $resultado;
   }
```

Tras esta la ejecución de esta función se devuelven los valores pedidos en orden inverso al seguido hasta el momento hasta llegar nuevamente a la vista, en caso de existir notificaciones, se muestra al usuario el siguiente mensaje:

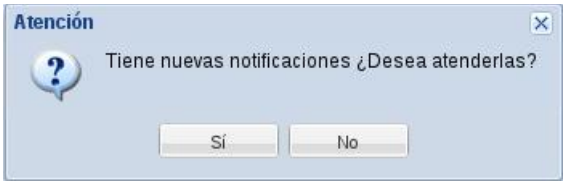

Figura No 12. Mensaje Mostrado al Usuario si se detectan notificaciones pendientes

Como conclusión del ejemplo se puede comentar que se ejecutan tareas específicas de cada capa en cada momento, cada una de estas acordes con la función de cada capa y cada elemento de ellas, proporcionando así una clara organización de cada elemento de la solución, lo que evidencia sencillez, agilidad y eficiencia en el proceso, así como posibilidades de escalabilidad, mantenimiento y modificaciones del sistema sin grandes costos.

3.8. Resumen la Fase de Implementación

La etapa de implementación resultó sin ninguna duda una de las más complejas durante el desarrollo de la solución, se obtuvieron un grupo de funcionalidades distribuidas en las distintas clases del negocio, se extendió el suAdminPlugin y el subsistema RC para lograr cumplir los nuevos requisitos especificados. Durante este proceso se consumieron varios servicios del subsistema Tablas de Control, y se crearon un grupo de servicios para otros subsistemas que necesitan información del RC y Administración para la correcta ejecución de algunas de sus funcionalidades.

3.9. Pruebas

Las pruebas constituyen el único instrumento adecuado para determinar el status de la calidad de una solución informática. Durante este proceso se ejecutan pruebas dirigidas a componentes del software o al sistema de software en su totalidad, con el objetivo de medir el grado en que este cumple con los requerimientos. Básicamente los objetivos fundamentales del proceso de pruebas son los siguientes:

- Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de integración y las pruebas de sistema. Las pruebas de integración son necesarias para la construcción dentro de la iteración, mientras que las pruebas de sistema son necesarias sólo al final de la iteración.
- Diseñar e implementar las pruebas creando los casos de pruebas que especifican qué probar, creando los procedimientos de prueba que especifican cómo realizarlas y creando, si es posible, componentes de pruebas ejecutables para automatizarlas.

- Realizar las diferentes pruebas y manejar los resultados de cada una sistemáticamente. Las construcciones en las que se detectan defectos son probadas de nuevo y posiblemente devueltas a otro flujo de trabajo, como diseño o implementación, de forma que las no conformidades puedan ser corregidas. (43)

En el presente epígrafe se hace una reseña del proceso de pruebas aplicado a la solución desarrollada, iniciando con la argumentación de las diferentes técnicas de pruebas de software que existen.

3.9.1. Técnicas de Evaluación de Software

Uno de los elementos determinantes de la calidad de una solución informática, está determinada entre otros elementos, por la coincidencia entre lo concebido inicialmente y lo logrado en la primera fase. Para comprobar el grado de cumplimiento de estos requisitos se usan dos técnicas de pruebas de software las cuales se describen a continuación:

Técnicas de Evaluación Estáticas: buscan faltas sobre el sistema en reposo. Estudian los distintos modelos que componen la solución buscando posibles fallas. Así pues estas técnicas se pueden aplicar, tanto a requisitos como a modelos de análisis, diseño y código.

Técnicas de Evaluación Dinámicas: generan entradas al sistema con el objetivo de detectar fallos cuando el sistema se ejecuta recibiendo dichas entradas. Los fallos se observan cuando son detectadas incongruencias entre la salida esperada y la real. La aplicación de técnicas dinámicas es también conocida como pruebas de software o *testing* y se aplican generalmente sobre código (44). Estas técnicas se agrupan:

- Técnicas de caja blanca o estructurales, que se basan en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa.
- Técnicas de caja negra o funcionales, que realizan pruebas sobre la interfaz del programa a probar, entendiendo por interfaz las entradas y salidas de dicho programa. No es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar (44).

3.10. Estrategias de Pruebas de Caja Negra Aplicadas

Las estrategias de pruebas de Caja Negra consisten en los pasos a seguir cuando se evalúa dinámicamente una solución informática, comenzando desde los componentes más simples y sencillos hasta evaluar todo el software en su conjunto.

A continuación se muestra el diseño de casos de pruebas para el RF Gestionar Notificaciones, el cual se encuentra compuesto por varios escenarios los cuales agrupan cada una de las funcionalidades que dieron solución a dicho requisito, y el cual se toma de ejemplo para mostrar las pruebas realizadas a la solución.

Tabla 1: Diseño de casos de prueba para el RC Gestionar Notificaciones.

Nombre del Requisito	Descripción General	Escenarios de Pruebas	Flujo del Escenario
Gestionar Notificaciones	Requisito mediante el cual se gestionan las notificaciones enviadas a administración por el subsistema RC	EP 1.1 Mostrar Notificaciones.	• Se muestra una interfaz con los datos de las notificaciones existentes.
		EP 1.2 Procesar Notificación	• Se selecciona la notificación que se desea procesar de un listado de estas. • Al seleccionar la notificación a procesar, se llena automáticamente el formulario de gestión de notificaciones. • Se hacen los cambios necesarios en los campos del formulario y se llenan los no autogenerados. • Se pulsa el botón “Procesar Notificación” y en caso de ser correctos los datos, estos son guardados en la base de datos, llevándose a cabo automáticamente la acción pertinente en la gestión de usuarios, en caso de los datos ser incorrectos se muestra el mensaje al usuario.

3.10.1. Pruebas Unitarias

Las pruebas unitarias son una estrategia de prueba de caja negra. Estas aseguran que un único componente de la aplicación produce una salida correcta para una determinada entrada. Este tipo de pruebas validan la forma en la que las funciones y métodos trabajan en cada caso particular. Las pruebas unitarias se encargan de un único caso a la vez, lo que significa que un único método puede necesitar varias pruebas unitarias en dependencia de su funcionamiento en cada contexto.

Las pruebas unitarias del *framework Symfony* son archivos PHP normales cuyo nombre termina en *Test.php* y que se encuentran en el directorio *test/unit* de la aplicación. Su sintaxis es sencilla y fácil de implementar.

3.10.1.1. Aplicación de Pruebas Unitarias a la Solución

A continuación se muestran las pruebas unitarias del *framework Symfony* aplicadas al escenario de prueba 1.1 (Mostrar Notificaciones) descrito en el Diseño de casos de prueba para el RF Gestionar Notificaciones (Tabla 1).

EP 1.1 Mostrar Notificaciones.

Esta funcionalidad debe retornar las notificaciones existentes en la base de datos.

Tabla 2: Resultados esperados y obtenidos durante la aplicación de las pruebas unitarias a este cado escenario.

No	Descripción	Resultado Esperado	Resultado Obtenido
1	Tipo de Resultado	Array	Array
2	Cantidad Notificaciones	1	1
3	Datos de Notificación	“IdNotif”: 22 “IdTipoTrabajador”: 1 “CodigoTipoTrabajador”: DECLARANTE “IdPersona”: 81154 “CI”: 88111600501 “PrimerNombre”: LEODAN “PrimerApellido”: RODRIGUEZ “SegundoApellido”: DIAZ “CodigoTrabajadorExterno”: TE_4fc06202b7340	“IdNotif”: 22 “IdTipoTrabajador”: 1 “CodigoTipoTrabajador”: DECLARANTE “IdPersona”: 81154 “CI”: 88111600501 “PrimerNombre”: LEODAN “PrimerApellido”: RODRIGUEZ “SegundoApellido”: DIAZ “CodigoTrabajadorExterno”: TE_4fc06202b7340

A continuación se muestra la implementación de esta prueba en el *framework Symfony* :

```
$notificaciones  = notificacionesPendientes();

$resPrueba->isa_ok($notificaciones, 'array')
$resPrueba->is($result[0]['idNotif'], 23);
$resPrueba->is($result[0]['idTipoTrabajador'], 1);
$resPrueba->is($result[0]['codigoTipoTrabajador'], 'DECLARANTE');
$resPrueba->is($result[0]['idPersona'], 81154);
$resPrueba->is($result[0]['CI'], 88111600501);
$resPrueba->is($result[0]['PrimerNombre'], 'LEODAN');
$resPrueba->is($result[0]['PrimerApellido'], 'RODRIGUEZ');
```

```
$resPrueba->is($result[0]['SegundoApellido'], 'DIAZ');
$resPrueba->is($result[0]['CodigoTrabajadorExterno'], 'TE_4fc06202b7340');
```

Y tras su ejecución se muestra la siguiente respuesta

```
www\GINA>symfony test: unit MostrarNotificaciones
1..10
ok 1
ok 2
ok 3
ok 4
ok 5
ok 6
ok 7
ok 8
ok 9
ok 10
Looks like everything went fine.
```

Como se puede apreciar en el ejemplo anterior se obtienen los resultados esperados en las pruebas unitarias. Cabe destacar que cada línea prueba un único elemento en todo el entorno de prueba, lo que posibilita detectar con mayor eficiencia y especificidad cada posible error. De esta misma forma fueron realizadas las pruebas al resto de los escenarios, obteniéndose, en la mayoría de los casos, los resultados esperados.

3.10.2. Pruebas Funcionales

Las pruebas funcionales constituyen una estrategia de evaluación de software. El objetivo fundamental de estas pruebas es validar si el comportamiento observado del software cumple o no con sus especificaciones definidas. La prueba funcional toma el punto de vista de usuario. Las funciones son probadas ingresando las entradas y examinando las salidas. La estructura interna de la solución es raramente considerada.

3.10.2.1. Aplicación de Pruebas Funcionales a la Solución

A continuación se muestran las pruebas funcionales aplicadas al escenario de prueba 1.2 (Procesar Notificaciones) descrito en el diseño de casos de prueba para el RF Gestionar Notificaciones (Tabla 1).

Tabla 3. Juego de datos para el entorno de pruebas 1.2 (Procesar Notificaciones) descrito en el Diseño de casos de prueba para el RF Gestionar Notificaciones (Tabla 1).[11]

Id EP	EP	Acción	Usuario	Contra.	ID Pers.	Tipo Trab.	Respuesta Esperada	Respuesta del Sistema
1.2	Procesar Notif.	V	V	V	V	V"	Se guardan los datos en la BD y se muestra un mensaje informativo al usuario.	Se guardan los datos en la BD y se muestra un mensaje informativo al usuario.
		I	V	V	I	I	Se lanza una excepción y se muestra el mensaje de error al usuario.	Se lanza una excepción y se muestra el mensaje de error al usuario.

3.11. Resumen de la Fase de Pruebas

Pruebas de Caja Negra como las ejemplificadas anteriormente fueron aplicadas al resto de los requisitos. Las funcionalidades fueron divididas en un total de 35 entornos de prueba, y en estos fueron detectadas 3 no conformidades, para más de un 91 % de aceptación, como se puede observar en la siguiente gráfica:

Figura No 13. Gráfico de los resultados de las pruebas

Posteriormente a este ciclo de pruebas, fueron corregidas las no conformidades detectadas, lo que posibilitó que no se detectaran no conformidades en ningún entorno de pruebas durante la segunda iteración de pruebas.

[11] Las celdas de la tabla contienen V(válido)o I(inválido)

Las pruebas fueron realizadas por un equipo de analistas del subsistema de Administración y Configuración del Departamento de Soluciones para la Aduana del CEIGE.

3.12. Aporte y Novedad de la Solución

Con la obtención de la solución propuesta, se logró mejorar considerablemente varias funcionalidades de los subsistemas Registro Central y Administración del sistema GINA, así como se desarrollaron nuevas funcionalidades necesarias para un completo y correcto funcionamiento de estos subsistemas, contribuyendo a un buen desempeño del sistema y evidentemente de la AGR como usuario del mismo; logrando un gran paso hacia la independencia tecnológica del país.

Al hacer uso de herramientas y tecnologías libres, así como patrones y estándares definidos y bien recomendados, fue posible una solución precisamente libre y completamente adaptable al sistema GINA y a los demás subsistemas que ya se encuentran en explotación y de los que se encuentran actualmente en desarrollo.

Según valoraciones del personal aduanero y especialistas de CADI, al valorar la solución, exponen que se agiliza significativamente los procesos de interacción de entidades y trabajadores externos con la AGR al informatizarse estos, además comentan que se logra el acceso a información necesaria desde cualquier punto autorizado y que se mantiene un historial de todos los trabajadores, entidades y sus cambios en el tiempo, elemento de vital importancia si son necesitados posteriormente.

3.13. Conclusiones Parciales

La anterior etapa de diseño y sus artefactos, posibilitaron una ejecución evidentemente viable del proceso de implementación. Se evidenciaron además las muchas facilidades del framework Symfony para desarrollar aplicaciones web dinámicas de gestión. El estudio de las Técnicas de Evaluación de Software, permitió que se profundizara en el uso de las pruebas de Caja Negra aplicadas a una solución informática, y específicamente en el caso de la presente, posibilitó la detección de varias no conformidades, permitiendo su corrección y la evidente mejora del sistema hasta hacerlo totalmente funcional.

Conclusiones

El presente trabajo, ha posibilitado dar solución a un grupo de necesidades del RCE de la AGR, así como de necesidades de seguridad de los sistemas y la información que maneja la AGR cuando interactúa con agentes externos. Dicha solución permite la gestión automatizada de la información de entidades y sus trabajadores que interactúan con la AGR, así como les permite en dependencia de sus permisos consultar dicha información en los casos necesarios.

Para la conformación de la propuesta de solución se realizó un estudio del estado del arte, y se tomaron algunas experiencias para mejorar los sistemas existentes y en uso en la AGR, así como desarrollar las nuevas funcionalidades basándose en dichas experiencias. Se hizo uso además del framework de PHP Symfony, para el desarrollo de la lógica del negocio, Propel para el acceso y mapeo de los datos y para la capa de interfaz de usuarios el framework de Java Script, ExtJs.

Por lo anteriormente expuesto, se concluye que los objetivos propuestos para el presente trabajo han sido cumplidos satisfactoriamente, ya que la aplicación da solución a la situación problémica que le dio origen.

Recomendaciones

A pesar de que se hayan cumplido correctamente los objetivos trazados en este trabajo, durante su solución surgieron nuevas ideas, las cuales se recomienda tener en cuenta:

- Continuar desarrollando nuevas funcionalidades que aunque no sean críticas, hagan a la solución más competitiva, eficiente y fácil de usar por el usuario.
- Continuar el estudio de los procedimientos en Registros Centrales con motivo de mejorar su entendimiento y lograr mejores propuestas de automatización.
- Continuar el estudio de los sistemas de administración y gestión de acceso, para perfeccionar en lo posible las soluciones existentes en el GINA.
- Continuar el seguimiento de las actualizaciones de las herramientas y tecnologías usadas para garantizar mejoras en versiones futuras de la solución, además de extender la búsqueda de nuevas tecnologías que en lo posibles puedan adoptarse y perfeccionar la solución.

Referencias Bibliográficas

1. **AGR.** SITIO WEB DE LA ADUANA CUBANA. [En línea] AGR. [Citado el: 10 de diciembre de 2011.] http://www.aduana.co.cu.

2. **Pérez, Pedro Ramón Pupo.** Resolución No 807-2009 de la Aduana General de la República. La Habana : s.n., 2009.

3. *Revista Pedagogía Universitaria.* **García, Dr. C. María Elena Guardo.** 3, Matanzas : Facultad Cultura Física, 2009, Vol. XIV.

4. **Otros, Mikael Berndtsson y.** Thesis Projects: A Guide for Students in Computer Science. London : s.n., 2002. 978-1-84800-008-7. Springer-Verlag.

5. *Real Academia de la Lengua Española.* [En línea] RAE. [Citado el: 10 de 2 de 2012.] http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=aduana.

6. **INSTITUTE, I. G. COBIT 4.0. 2005, Disponible en: www.itgi.org. ISBN 1-933284-37-4.** [En línea]

7. **M. GERBER, R. V. SOLMS.** Formalizing information security requirements. Disponible en http://www.emeraldinsight.com/10.1108/09685220110366768 : s.n., 2001. Vol. 9. ISBN 0968-5227.

8. **C. GUTIÉRREZ, E. FERNÁNDEZ-MEDINA.** A Survey of Web Services Security. Springer Berlin / Heidelberg : s.n., 2004. Vol. 3043/2004, Disponible en: http://www.springerlink.com/. ISBN 978-3-540-22054-1.

9. **López, M. J.** *Criptografía y Seguridad en Computadores.* s.l. : 3ra Edición, 2003.

10. **A. J. MENEZES, P. OORSCHOT.** Handbook of Applied Cryptography. s.l. : Disponible en: http://www.cacr.math.uwaterloo.ca/hac/, 1996. ISBN 0-8493-8523-7.

11. **A. J. STELL, D. R. O SINNOTT.** Comparison of Advanced Authorisation Infrastructures for Grid Computing. s.l. : 1ra ed. IEEE, 2005. Vol. Disponible en: http://eprints.gla.ac.uk/3560/. ISBN 1550-5243.

12. ARGENTINA, C. C. N. CCN -CERT Argentina. s.l. : Disponible en: https://www.ccn-cert.cni.es/.

13. **TONG, E. YUAN y J.** Attributed Based Access Control (ABAC) for Web Services. *En IEEE International Conference on Web Services (ICWS'05).* 2005.

14. **SANDHU, R. S.** Lattice-Based Access Control Models IEEE. 1993. ISBN 0018-9162/93/1100-0009.

15. Sitio web de la Ventanilla Única de Comercio Exterior de Perú. [En línea] [Citado el: 5 de febrero de 2012.] https://www.vuce.gob.pe/.

16. **Tamayo, Islandy Antonio Guevara.** *Análisis y Diseño del Modulo Tablas de Control del SUA.* UCI La Habana : s.n.

17. **IEEE.** IEEE Standard Glossary of Software Engineering Terminology. [En línea] 1990. [Citado el: 5 de marzo de 2012.] http://standards.ieee.org/findstds/standard/610.12-1990.html.

18. *La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software.* **Chávez, Michael Arias.** Universidad de Costa Rica : Revista InterSedes : s.n., 2005, Vol. VI. ISSN 1409-4746.

19. *Ingeniería de Software: Un enfoque práctico.* **Pressman, Roger.** México DF : s.n., 2006.

20. *Ingeniería de Requisitos en Aplicaciones para la Web - Un estudio comparativo.* **María José Escalona, Nora Koch.** Universidad de Sevilla : Departamento de Lenguajes y Sistemas Informáticos : s.n., 2002.

21. **Escalona, MJ. y Koch, N.** *Ingeniería de Requisitos en Aplicaciones para la Web: Un estudio comparativo. .* España y Alemania : s.n., 2002.

22. **Sommerville, Ian.** *Ingenieria del Software.* Enero, 2005.

23. **Reyero, Eusebio.** Metodologías de diseño | Will Web For Food. [En línea] 2008. [Citado el: 3 de marzo de 2012.] http://wwff.thespacer.net/blog/metodologias-de-diseno/.

24. **Billy, Carlos Reynoso.** *Introducción a la Arquitectura de Software.* Buenos Aires : Universidad de Buenos Aires : s.n., 2004.

25. **Clements, Paul C.** *Software Architecture.* Estados Unidos : Software Engineering Institute : s.n., 2002. ISBN: 15213-3890..

26. **Hernández, Pedro Veloso.** *Uso de Patrones de Arquitectura.* 2009.

27. **Pantoja, Ernesto Bascón.** *El patrón de diseño modelo -vista - controlador (MVC) y su implementación el Java Swing.* 2004.

28. **Visual Paradigm International.** UML, BPMN and Database Tool for Software Development. [En línea] [Citado el: 3 de marzo de 2012.] http://www.visual-paradigm.com.

29. **Pérez, Javier Eguíluz.** *Introducción a XHTML.* 2007.

30. —. *Introducción a CSS.* 2008.

31. —. *Introducción a JavaScript.* 2007.

32. **The PHP Documentation Group.** *Manual de PHP.* 2011.

33. Programación en Castellano. *¿Por qué elegir PHP?* [En línea] [Citado el: 26 de febrero de 2012.] http://www.programacion.com/articulo/por_que_elegir_php_143.

34. **Cobo, Jose A.** *Línea Base Arquitectónica para el Polo Sistemas Tributarios y de Aduanas.* 2008.

35. **Comunidad de Desarrollo de Ext JS.** Ext JS en Español. [En línea] [Citado el: 1 de marzo de 2012.] http://extjs.es/.

36. **Potencier, Fabien y Zaninotto, François.** *Symfony la guía definitiva.* 2008.

37. **Escuela de Ingeniería Informática Universidad de Oviedo.** Entornos de Desarrollo Integrado. [En línea] 2008. [Citado el: 7 de marzo de 2012.] http://petra.euitio.uniovi.es/~i1667065/HD/documentos/Entornos%de%Desarrollo%Integrado.pdf.

38. **Oracle Corporatio.** Portal del IDE Java de Código Abierto. [En línea] 2010. [Citado el: 12 de marzo de 2012.] http://netbeans.org/index_es.html.

39. **CAVSI.com.** ¿Qué es un Sistema Gestor de Bases de Datos o SGBD? [En línea] 2007. [Citado el: 6 de marzo de 2012.] http://www.cavsi.com/preguntasrespuestas/que-es-un-sistema-gestor-de-bases-de-datos-o-sgbd/.

40. **Martínez, Ing. Jenni Manso.** *Procedimiento para la Ingeniería de Requisitos en el Departamento de Desarrollo de Soluciones para la Aduana del CEIGE.* La Habana : Universidad de las Ciencias Informáticas : s.n., 2010.

41. **Ivar Jacobson, Grady Booch y James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 1999. ISBN: 0-201-57169-2.

42. **Daniuska Laurencio Pérez, Yunieski Diéguez García.** *Sistema de Gestión de la Trayectoria Productiva de cada estudiante en la facultad 1.* Universidad de las Ciencias Informáticas La Habana : s.n., 2009.

43. **Ivar Jacobson, Grady Booch y James Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley. ISBN: 0-201-57169-2.

44. **Natalia Jurista, Ana M. Moreno y Sira Vegas.** *Técnicas de Evaluación de Software.* 2006.

45. MasterMegazine. [En línea] [Citado el: 8 de febrero de 2012.] http://www.mastermagazine.info/termino/4908.php.

46. **Javier Llácer Muñoz, Carlos Martínez Burgos y Sergio Catalá Gil.** *Informe de evaluación de ERP.*

47. **Innova., Grupo Soluciones.** [En línea] Rational Unified Process. [Citado el: 3 de Marzo de 2012.] http://www.rational.com.ar/herramientas/rup.html.

48. **Larman, Craig.** *UML y Patrones.* 1999. ISBN 970-1 7-0261-1.

49. **Fowler, Martin.** *Patterns of Enterprise Application Architecture.* ISBN: 0-32112-742-0.

50. **Rodriguez, Jose Antonio Cobo.** *Línea Base Arquitectónica para el Polo Sistemas Tributarios y de Aduanas.* 2008.

51. Aduana de Cuba. [En línea] 2008. [Citado el: 8 de 12 de 2011.] www.aduana.co.cu.

52. VUCE Perú. [En línea] 2010. [Citado el: 8 de 12 de 2010.] www.vuce.gob.pe.

53. *Recomendacion No. 33 UN/CEFACT Naciones Unidas para la Facilitación del Comercio y el Negocio Electrónico.* **UN/CEFACT.** 2004.

54. VUCEM. [En línea] 2011. [Citado el: 7 de 12 de 2011.] www.ventanillaunica.gob.mx/vucem.

55. **Klimovsky, Gregorio.** Las desventuras del conocimiento científico. s.l. : A-Z editora, 1997.

56. Ventanilla Única de Suecia. [En línea] 2011. www.tullverket.se.

57. ITDS Ventanilla Única EEUU. [En línea] 2011. www.itds.gov.

58. **Garlan, D y Shaw, M.** *Software Architecture: Perspectives on an Emerging Discipline.* 1996.

59. **Perry, D y Wolf ,A.** *Foundations for the Study of Software Architecture.* 1992.

60. **A., CHRISTOPHER.** *A Pattern Language: Towns, Buildings, Construction.* 1977.

61. **Buschmann, F.** *Pattern-Oriented Software Architecture: A System of Patterns.* 1996.

62. **Lutz, Mark.** *Learning Phython, 4th Edition.* 2010.

63. **Tucumán, Universidad Tecnológica Nacional-Facultad Regional.** Lenguajes de Programación. [En línea] 2000. [Citado el: 15 de Enero de 2011.]

64. PHP.net. [En línea] 2001. [Citado el: 15 de 1 de 2011.] php.net/manual/en/faq.general.php.

65. **Potencier, Fabien.** *Symfony la guía definitiva.* 2008.

66. **Burker, Coyner B &.** *Extreme Programming.* 2005.

67. **Matraspa, Verdecia D I &.** *Procedimiento para el modelado de los procesos del Negocio y la Captura de Requisitos del Software a la medida desarrollados en la UCI.* 2008.

68. Rational. [En línea] 2007. www.rational.com.ar.

69. Buenas Tareas. [En línea] www.buenastareas.com/Metodologias-de-Desarrollo-de Software.

70. **UN/CEFACT.** *II Encuentro Regional Latinoamericano y del Caribe sobre las VUCE.* 2010.

71. **Falgueras, Benet Campderrich.** *Ingenieria del Software.* 2003.

72. **Menendez, Abg Juan Jose Gaviria.** *Ventanilla Única Ecuatoriana.* s.l. : Federacion Ecuatoriana de Agentes de Aduana.

73. The Korea International Trade Association. [En línea] 2000. [Citado el: 14 de 2 de 2012.] http://www.kita.net.

74. Unipass-Korea Custom Services. [En línea] 2009. [Citado el: 14 de 2 de 2012.] http://www.unipass.or.kr.

75. **Bizagi.** bizagi. [En línea] 2011. [Citado el: 2 de 3 de 2012.] http://www.bizagi.com/docs/BPMNbyExampleSPA.pdf.

76. **Group, Object Management.** bpmn. [En línea] omg, 1997. [Citado el: 2 de 3 de 2012.] http://www.bpmn.org/.

77. **Beatriz Ayala, Claudia Marcela Ramírez, Lina María Ocampo.** *La Ingeniería de Requerimientos aplicada al desarrollo de sistemas de información.* Manizales,Colombia : s.n., 2011.

78. **Driggs, Idalberto Carlos Alonso.** *Modelo de Desarrollo del Departamento de Soluciones para la Aduana.* Universidad de las Ciencias Informáticas : s.n., 2012.

79. eZ Component. [En línea] 2007. eZComponents.org.

80. ECURED. [En línea] 16 de Abril de 2012. www.ecured.cu.

81. **Laurencio Pérez, Daniuska y Diéguez García, Yunieski.** *Sistema de Gestión de la Trayectoria Productiva de cada estudiante en la facultad 1.* Universidad de las Ciencias Informáticas La Habana : s.n., 2009.