Universidad de las Ciencias Informáticas

Facultad3

Título: Propuesta de un mecanismo de seguridad para el marco de trabajo Kairos

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor(es):

- Lenis Matos Rodríguez
- Daniel Sarmiento Sastriques

Tutor(es):

- Julio César Prieto Álvarez

Co-Tutor(es):

- Alberto Jesús La Rosa Agramonte

Ciudad de La Habana

Junio 2012

"Saber no es suficiente, debemos aplicar. Desear no es suficiente, debemos hacer."

Johann Wolfgang von Goethe.

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Daniel Sarmiento Sastriques	**Lenis Matos Rodríguez**	**Julio C. Prieto Álvarez**
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

Datos de contacto

Agradecimientos

Dedicatoria

Resumen

Gran parte de las soluciones de software que se desarrollan en la Universidad de las Ciencias Informáticas son de corte empresarial. Muchas de ellas creadas como aplicaciones distribuidas, utilizando la plataforma Java, en especial la conocida como plataforma Edición Empresarial de Java. Para facilitar el desarrollo de estas se ha propuesto la creación del marco de trabajo Kairos que permitirá la reutilización de los componentes necesarios en este tipo de sistemas, además proporciona a las aplicaciones que lo utilicen funcionalidades y componentes que agilizan y simplifican su desarrollo, como son la gestión de las sesiones de los usuarios, la gestión de trazas y excepciones y otras funcionalidades de administración del sistema. Uno de los elementos fundamentales con que debe contar este marco de trabajo es la seguridad, por lo que esta investigación estará enfocada principalmente en el desarrollo de los mecanismos de seguridad necesarios para Kairos.

Para el desarrollo de la solución se analizaron varias herramientas y tecnologías seleccionando las más adecuadas con respecto a las características del sistema, validando el mismo por medio de las pruebas de unidad y de aceptación.

Índice

Declaración de autoría ..... *I*

Datos de contacto ..... *II*

Agradecimientos ..... *III*

Dedicatoria ..... *IV*

Resumen ..... *V*

Introducción *9*

Capítulo 1: Fundamentación teórica *13*

1.1 Introducción **13**

1.2 Seguridad lógica de la información **13**

1.3 Algoritmos de encriptación **14**

1.3. 1 Algoritmos de resumen 14

1.3.1.1 Md5 15

1.3.1.2 Sha-1 15

1.3. 2 Algoritmos de cifrado simétrico 15

1.3.2.1 Triple-DES 16

1.4 Aplicaciones distribuidas **16**

1.4. 1 Problemas de seguridad en las aplicaciones distribuidas 17

1.5 El marco de trabajo Kairos **17**

1.6 Metodologías de desarrollo **18**

1.6. 1 Proceso Unificado de Desarrollo 19

1.6. 2 Programación Extrema 20

1.7 Herramientas CASE **23**

1.7.1 Visual Paradigm for UML v3.4 23

1.7.2 IBM Rational Rose Enterprise Edition v7.0 24

1.8 Lenguaje de modelado **24**

1.8.1 Lenguaje Unificado de Modelado v2.0 24

1.9 Entornos de Desarrollo Integrado **25**

1.9. 1 NetBeans v7.0.1 25

1.9. 2 Eclipse Índigo v3.7 25

1.10 Servidor de aplicaciones 26
1.10. 1 Dominios de seguridad en el GlassFish Open Source Edition v3.0.1 26
1.10. 2 EJB v3.0 27
1.11 Sistemas Gestores de Bases de Datos 28
1.12. 1 PostgreSQL v8.4 28
1.12. 2 Oracle DB Edición Empresarial 29
1.12 Pruebas de unidad 29
1.13 Conclusiones del capítulo 30
Capítulo 2: Descripción y diseño de los mecanismos de la propuesta de solución 31
2.1 Introducción 31
2.2 Propuesta de solución 31
2.3 Diseño de la solución 32
2.3.1 Patrones GRASP 32
2.3.2 Patrones GoF 33
2.3.3 Tarjetas CRC 35
2.3.4 Arquitectura de la solución 38
2.3.5 Diagramas de clases 40
2.4 Definición de los mecanismos de seguridad 43
2.4.1 Mecanismo de autenticación y autorización 43
2.4.1.1 Dominio de Seguridad 43
2.4.1.1.1 Configuración 44
2.4.1.2 Autenticación desde el cliente 46
2.4.1.3 Autorización en el servidor 46
2.4.2 Validación de la fortaleza de la contraseña 48
2.4.3 Encriptación de contraseña 49
2.4.4 Bloqueo de sesión por inactividad 49
2.4.5 Mecanismo de confidencialidad e integridad 50
2.4.5.1 Mecanismo de confidencialidad 50
2.4.5.2 Mecanismo de integridad 52
2.4.5.3 Abstracción 53
2.4.5.4 Encapsulamiento 54
2.4.6 Activación de estaciones de trabajo 55
2.5 Conclusiones 56
Capítulo 3: Implementación y pruebas 57
3.1 Introducción 57

3.2 Modelo de Implementación 57
3.2.1 Diagrama de Componentes 57
3.2.2 Diagrama de Despliegue 65
3.3 Estándares de codificación 67
3.4 Modelo de pruebas 69
3.4.1 Pruebas de unidad 69
3.4.2 Pruebas de aceptación 76
3.5 Conclusiones 76
Conclusiones generales 78
Recomendaciones 79
Bibliografía 81
Glosario de términos 84

Introducción

Desde finales del siglo pasado las Tecnologías de la Información y las Comunicaciones (TIC) se han venido consolidando como uno de los principales pilares en el desarrollo de la sociedad. La informática ocupa un lugar de gran importancia dentro de las TIC, estando presente prácticamente en todo el quehacer del hombre. Permite el almacenamiento, búsqueda y análisis de grandes volúmenes de información, posibilitando realizar tareas que eran impensables tiempo atrás, además es ampliamente aplicada en las ramas empresarial y gubernamental.

Desde hace algunos años Cuba viene desarrollando soluciones informáticas vinculadas con estas ramas, muchas de las cuales han sido creadas en la Universidad de las Ciencias Informáticas (UCI), donde existen varios centros de desarrollo de software adscritos a ella, como son el Centro de Gobierno Electrónico (CEGEL), y el Centro de Investigación y Gestión de Entidades (CEIGE), especializados en realizar soluciones dentro del contexto nacional, como parte de la política de informatizar la sociedad cubana. También se aplican en el contexto internacional como resultado del convenio de colaboración entre Cuba y la República Bolivariana de Venezuela.

Las soluciones de software empresariales actualmente son desarrolladas para brindar servicios a un amplio rango de usuarios, ya sea dentro del marco de una institución, un conjunto de instituciones o incluso a través de internet. Estas aplicaciones manejan gran cantidad de información, que constituye el bien más preciado de cualquier organización o empresa, como pueden ser datos de los clientes, de cuentas bancarias o del funcionamiento de la propia institución. Por este motivo se hace necesario que esta sea accedida, creada y modificada por las personas que cuenten con los permisos para ello. En caso contrario la organización correría el riesgo de que utilicen dicha información maliciosamente para obtener ventajas o que sea manipulada, ocasionando lecturas erradas o incompletas.

Actualmente se dedican grandes esfuerzos en el marco de las aplicaciones informáticas, en pos de alcanzar un nivel de protección acorde con la importancia de los datos que se manejan y las posibles amenazas que afectan dichas soluciones. Por esta razón uno de los elementos más importantes dentro de las soluciones de software empresariales es la seguridad, la cual requiere el empleo de una considerable cantidad de tiempo y esfuerzo, convirtiéndose en uno de los elementos más importantes en la concepción, desarrollo y puesta en marcha.

Entre las plataformas que se utilizan en CEGEL para el desarrollo de este tipo de aplicaciones se encuentra la plataforma Java y dentro de esta específicamente, la Edición Empresarial de Java (JEE por sus siglas en inglés), la cual cuenta con varias tecnologías que facilitan en gran medida el trabajo, como son la administración de transacciones, servicio de mensajería, un mecanismo para la invocación de procedimientos remotos y una Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) para la persistencia de datos.

Uno de los resultados alcanzados por la UCI utilizando esta tecnología es la creación del marco de trabajo Kairos, que cuenta con un conjunto de componentes que dotan a las aplicaciones informáticas que lo utilizan de funcionalidades indispensables, como son la gestión de usuarios, la gestión de las sesiones de los usuarios, el control de trazas, la internacionalización de las interfaces de usuarios y otras funcionalidades de administración del sistema. Kairos provee un alto nivel de configurabilidad y adaptabilidad, lo cual lo convierte en un software altamente reusable. Actualmente este marco de trabajo tiene grandes cualidades para la administración de las aplicaciones informáticas pero carece de elementos de seguridad que contribuyan al aseguramiento de la información, como son la encriptación de contraseñas, el control y configuración de la fortaleza que deben poseer las contraseñas del sistema, el tiempo de inactividad permitido antes de que la aplicación se bloquee, un mecanismo que permita realizar la autenticación de forma remota contra un servidor de aplicaciones, controlar las estaciones de trabajo que pueden acceder a una aplicación desplegada en el servidor y el intercambio de información de forma íntegra y confidencial.

Lo anteriormente planteado conlleva al siguiente **problema científico**, que sirve como línea base de la presente investigación. ¿Cómo contribuir al aseguramiento de la información gestionada por las aplicaciones de software que utilizan el marco de trabajo Kairos?

El **objeto de estudio** es la seguridad de la información en aplicaciones informáticas. La seguridad de la información para aplicaciones informáticas desarrolladas sobre tecnologías java constituye el **campo de acción** debido a que es el área donde se centra la investigación.

Este trabajo tiene como **objetivo general** desarrollar un mecanismo de seguridad que permita asegurar la información gestionada por las aplicaciones de software que utilizan el marco de trabajo Kairos.

Con esta investigación se defiende la siguiente **idea**: El desarrollo de un mecanismo de seguridad para el marco de trabajo Kairos contribuirá al aseguramiento de la información que se gestiona en las aplicaciones de software que lo utilizan.

Para darle cumplimiento al objetivo planteado en la presente investigación se definen los siguientes **objetivos específicos**:

1. Elaborar el marco teórico de la investigación.
2. Analizar las necesidades de seguridad a contemplar en la propuesta de solución.
3. Desarrollar los componentes de seguridad que den solución a las necesidades identificadas.
4. Validar la solución propuesta.

Las **tareas a cumplir** durante la investigación son las siguientes:

1. Estudio del estado del arte de las propuestas de mecanismos de seguridad para soluciones informáticas desarrolladas con tecnología java.

2. Definición de las metodologías, herramientas y patrones a utilizar para el desarrollo de los componentes necesarios.

3. Identificación de las necesidades de seguridad a incluir.

4. Identificación de los componentes necesarios para dar solución a las necesidades planteadas.

5. Diseño e implementación de los componentes que formarán parte del mecanismo propuesto.

6. Realización de las pruebas de unidad para cada componente desarrollado.

7. Realización de pruebas de integración en el marco de trabajo Kairos.

En el presente trabajo se hace uso de varios métodos científicos de investigación. A continuación se hace referencia a ellos:

Métodos teóricos

Inducción - Deducción: Ha permitido arribar a conclusiones para determinar cómo contribuir al aseguramiento de la información gestionada por las aplicaciones de software que utilizan el marco de trabajo Kairos.

Analítico - Sintético: Ha permitido analizar las teorías, documentos, artículos e informes permitiendo la extracción de los elementos más importantes que se relacionan con la seguridad de la información en aplicaciones informáticas.

Hipotético - Deductivo: Ha permitido la identificación de nuevas soluciones a partir de la idea que se defiende, las cuales se fomentaran con la culminación de la investigación.

Métodos empíricos

Observación: Es utilizado en todo el proceso de la investigación ya que este es el instrumento universal del científico, se realiza de forma consciente y orientada a un objetivo determinado. Este método se utilizó para obtener una caracterización detallada de las soluciones existentes para sistemas similares.

La presente investigación está estructurada de la siguiente forma.

Capítulo 1 Fundamentación Teórica: Este capítulo se centra en el análisis de los principales elementos teóricos que se utilizaron en cada una de las fases de la investigación centrando la atención en la seguridad de la información en las aplicaciones empresariales y las facilidades de distintas herramientas existentes para el desarrollo de proyectos similares en el mundo y la UCI.

Capítulo 2 Descripción y diseño de los mecanismos de la propuesta de solución: En este capítulo se proponen los mecanismos necesarios para dar solución a la problemática planteada, así como el diseño y la implementación de los componentes necesarios para la aplicación de los mecanismos.

Capítulo 3 Implementación y pruebas: En este capítulo se propondrá el Diagrama de Despliegue y el Diagrama de Componentes, que conformarán el Modelo de Implementación, además se exponen los estándares de codificación establecidos, así como las pruebas realizadas a la solución para asegurar su correcto funcionamiento.

Capítulo 1: Fundamentación teórica

1.1 Introducción

En el presente capítulo se realiza un análisis detallado de los principales elementos teóricos y conceptos a tener en cuenta para dar solución al problema planteado, así como un estudio de las tecnologías, metodologías de desarrollo y herramientas a utilizar para el desarrollo de la solución.

1.2 Seguridad lógica de la información

La seguridad lógica consiste en la aplicación de barreras y procedimientos para proteger la información de forma tal que sólo se permita el acceso a las personas que cuenten con la debida autorización. (1)

Existe información que puede ser pública, como es el índice de natalidad infantil en un país y otra que sólo puede ser visualizada por un grupo selecto de personas que trabajan con ella, como los antecedentes penales de un individuo. En este último caso se deben maximizar los esfuerzos para su preservación teniendo en cuenta que la información es indispensable para garantizar la continuidad operativa de cualquier institución, es un activo con valor en sí misma y debe ser conocida solo por las personas que la procesan.

Es por este motivo que se debe tener especial cuidado en mantener la integridad de la información la cual se puede ver afectada por anomalías en el hardware, software, virus informáticos y/o modificación por personas que se infiltran en el sistema. No menos importante es la privacidad o confidencialidad de la información pues las fallas de este tipo pueden provocar severos daños a su propietario. Uno de los principales mecanismos utilizados actualmente para prevenir esto son los controles de acceso, que pueden ser implementados en el sistema operativo, sobre las aplicaciones, en Bases de Datos, en un paquete específico de seguridad o en cualquier otro utilitario.

Específicamente constituyen una importante ayuda para proteger al software de la utilización o modificaciones no autorizadas, para mantener la integridad de la información restringiendo la cantidad de usuarios con acceso permitido y para resguardar la información de accesos no autorizados.

Asimismo es conveniente tener en cuenta otras consideraciones referidas a la seguridad lógica, como son las relacionadas con el procedimiento para determinar si un permiso de acceso corresponde a un determinado recurso, como son:

- ✓ **Identificación y autenticación:** Es la primera línea de defensa de la mayoría de los sistemas computarizados que permite la prevención del ingreso de personas sin autorización. Es la base para la mayoría de los controles de acceso y para el seguimiento de las actividades de los usuarios, de manera que:
  - La identificación es el momento en que el usuario se da a conocer en el sistema.
  - La autenticación es la acción de verificar si un usuario es quien dice ser utilizando las credenciales provistas en la identificación.

  (2)
- ✓ **Autorización:** Es el procedimiento de comprobar si un usuario previamente autenticado cuenta con los permisos necesarios para acceder a una determinada información o funcionalidad que está solicitando. Uno de los mecanismos más utilizados para garantizar la autorización es la asignación de roles a los usuarios. Un rol puede ser visto como un permiso asociado a un usuario que le posibilitará realizar alguna acción en un sistema. Los roles pueden ser asignados a grupos de usuarios o a uno individual. (3)

1.3 Algoritmos de encriptación

Los sistemas de software en muchas ocasiones manejan información que solo debe ser conocida por las personas con la debida autorización ya que comprometerían su seguridad o supondría daños a los propietarios de la misma. Es por esto que se necesitan estos algoritmos, los que proporcionan la seguridad necesaria para disminuir los riesgos antes planteados.

1.3. 1 Algoritmos de resumen

Los algoritmos de resumen tienen aplicaciones importantes en las soluciones de software, especialmente para el desarrollo de esta investigación, entre las que se encuentran la encriptación de las contraseñas de

las cuentas de usuario y la implementación de mecanismos para la verificación de la integridad de la información.

1.3.1.1 Md5

Es un algoritmo de reducción o resumen capaz de procesar cadenas de caracteres de una longitud arbitraria, produciendo una salida de 128 bits. Fue diseñado por el profesor Ronald Rivest del Instituto de Tecnología de Massachusetts en el año 1991. Md5 es de dominio público y se utiliza para encriptar las contraseñas en varios sistemas como Unix y GNU/Linux. (4)

1.3.1.2 Sha-1

Es un algoritmo de reducción o resumen que puede procesar cadenas de hasta 2 a la 64 bits y produce una salida de 160 bits. Se basa en principios similares a los empleados en el algoritmo md5 aunque es mucho más robusto. Sha-1 es de dominio público y ha sido adoptado como el estándar para el procesamiento de información federal por los Estados Unidos de América. Además es muy utilizado para la encriptación de las contraseñas en múltiples sistemas. (5)

La presente investigación soporta ambos algoritmos, los cuales son empleados en muchos sistemas de software, además la plataforma Java brinda soporte para la utilización de ambos. La utilización de uno u otro es configurable permitiendo a quién utilice este mecanismo de seguridad seleccionar el más adecuado a sus necesidades.

1.3. 2 Algoritmos de cifrado simétrico

Los algoritmos de cifrado simétrico son de gran importancia para esta investigación ya que proporcionan el medio para que la información que se envía entre cliente y servidor pueda viajar de forma cifrada manteniendo su confidencialidad. Tanto emisor como receptor utilizan la misma llave para el cifrado y descifrado de la información de forma que ambos deben conocer de antemano la llave a utilizar.

1.3.2.1 Triple-DES

Para realizar el cifrado de la información que se transmite entre cliente y servidor esta solución contará con el algoritmo Triple-DES, que es una ampliación del algoritmo DES ya que este utiliza una llave de 56 bits, la cual no es suficientemente grande como para garantizar la seguridad contra un ataque de fuerza bruta. Se basa en tres pasadas de este último sobre los datos utilizando dos o tres llaves distintas, extendiendo así la longitud efectiva de la llave a 192 bits.

1.4 Aplicaciones distribuidas

Debido a que la presente investigación va dirigida a mejorar la seguridad en aplicaciones distribuidas es importante el análisis de las mismas. Una aplicación distribuida es aquella cuyo objetivo final se alcanza mediante la ejecución de diversos procesos independientes que por lo general se ejecutan en equipos diferentes y que de una forma u otra se pasan datos entre ellos mediante protocolos de comunicaciones bien establecidos. (6)

Generalmente las aplicaciones empresariales que se desarrollan en CEGEL son distribuidas basadas en una arquitectura Cliente-Servidor. En este tipo de soluciones de software los procesos están distribuidos en distintas capas, no solo lógicas sino que pueden ejecutarse en entornos separados, por ejemplo, en las estaciones clientes se ejecuta la lógica de presentación que se comunica a través de la red con la lógica de negocio que se ejecuta en una estación servidora remota. *Ver Fig. 1.*

Fig. 1 Arquitectura Cliente-Servidor.

1.4. 1 Problemas de seguridad en las aplicaciones distribuidas

En este tipo de aplicaciones los problemas de seguridad son mayores debido a que existen diversos componentes que se ejecutan en entornos separados y que interactúan a través de la red, lo que aumenta su vulnerabilidad.

Entre los problemas que se presentan se encuentra el de lograr que la información solo pueda ser consultada por las personas que cuenten con la autorización para ello, además la misma puede ser modificada por personas no autorizadas y luego ser colocadas en la red para continuar su camino. El proceso de autenticación y autorización de los usuarios es más complejo, pues se deben transmitir las credenciales hacia el servidor por medio de un canal inseguro como la red, lo que requiere enmascarar las contraseñas de las cuentas de usuario por medio de técnicas criptográficas para evitar que sean conocidas por personal externo. Es por esto que se ve la necesidad de que el marco de trabajo Kairos cuente con los mecanismos que permitan solucionar dichos problemas.

1.5 El marco de trabajo Kairos

El marco de trabajo Kairos proporciona a las aplicaciones que lo utilicen funcionalidades y componentes que agilizan y simplifican su desarrollo. El mismo está diseñado para la creación de aplicaciones con arquitectura Cliente-Servidor utilizando la plataforma JEE. Por esta razón Kairos consta de dos partes, una que se encuentra en el lado del cliente facilitando la creación de la capa de presentación y la comunicación con los objetos remotos desplegados en el servidor y una segunda que se encuentra en el lado del servidor donde se encuentra la lógica de negocio. *Ver Fig. 2.*

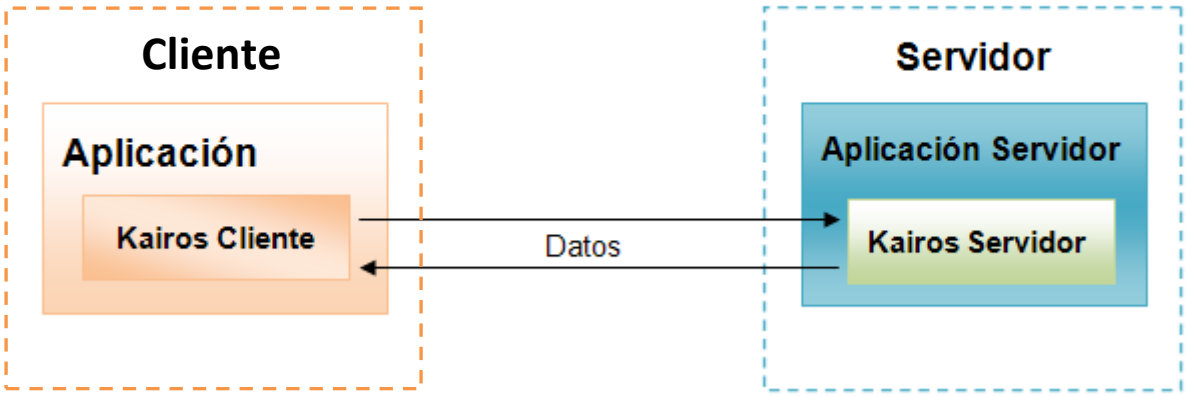


Fig. 2 Arquitectura de las aplicaciones que utilicen el marco de trabajo Kairos.

Algunas de las funcionalidades y componentes que provee son:

- Una interfaz de usuario básica.
- Un mecanismo para la internacionalización de las interfaces de usuario.
- Un mecanismo para el acceso a los procedimientos remotos publicados en un servidor de aplicaciones.
- Un mecanismo de mensajería.
- Un mecanismo para gestionar las excepciones lanzadas por la aplicación que lo utilice.

1.6 Metodologías de desarrollo

Desarrollar software es una actividad compleja que implica gran cantidad de riesgos. Con el devenir de los años y la experiencia adquirida de los éxitos y fracasos fueron surgiendo las diferentes metodologías de desarrollo de software, cuyo objetivo fundamental es el de guiar dicha actividad.

En aras de encontrar una metodología que se ajuste a las necesidades de esta investigación se analizaron dos de las más utilizadas a nivel mundial. Estas son el Proceso Unificado de Desarrollo (RUP por sus siglas en inglés) y Programación Extrema (XP por sus siglas en inglés).

1.6. 1 Proceso Unificado de Desarrollo

RUP es una metodología de desarrollo creada por la Corporación Rational Software, una de las más empleadas en el análisis, la implementación y la documentación de sistemas orientados a objeto. No es solamente un proceso normativo concreto, sino más bien un marco de trabajo adaptable, que le permite a las organizaciones desarrolladoras de software y a los equipos de desarrollo seleccionar los elementos del proceso que son apropiados para sus necesidades. (7)

Entre sus principales características podemos encontrar que:

- Está centrado en la arquitectura y guiada por los casos de uso, ofreciendo una forma disciplinada de asignar tareas y responsabilidades.
- Pretende implementar las mejores prácticas de la Ingeniería de Software.
- Promueve el uso de una arquitectura basada en componentes lo que facilita la reutilización de los mismos.
- Permite mitigar los posibles riesgos desde las etapas tempranas del desarrollo.

Esta metodología propone cuatro fases en las que se realizan distintos flujos, cada uno con más o menos fuerza dependiendo de la fase en que se encuentre. Los flujos propuestos son nueve en total, seis de ingeniería y tres de soporte. (8)

Flujos de ingeniería:

- *Modelado de negocio:* se lleva a cabo una descripción del negocio, definiéndose qué actividades se deben automatizar y qué personal debe participar en ellas.
- *Levantamiento de Requisitos:* se define lo que debe hacer el sistema de acuerdo a las funcionalidades deseadas por el usuario y las restricciones intrínsecas en esto.

- *Análisis y Diseño:* se describe cómo se deben cumplir las funcionalidades y restricciones descritas, detallando minuciosamente lo que debe ser programado.
- *Implementación:* se comienza con la implementación del sistema en término de clases y objetos.
- *Prueba:* Se identifican y eliminan los errores que puedan haber surgido durante el proceso de desarrollo.
- *Despliegue:* se pone en mano de los usuarios finales una liberación del producto.

Flujos de soporte:

- *Gestión de configuración y cambios:* para mantener a todo el equipo al tanto de las últimas versiones, se describe cómo controlar los artefactos generados en el proyecto.
- *Gestión de proyecto:* recoge las actividades que se deben realizar con vista a obtener un producto que satisfaga realmente las necesidades del cliente.
- *Gestión de ambiente:* contiene actividades que describen las herramientas que utilizará el equipo de trabajo.

1.6. 2 Programación Extrema

XP es una metodología de desarrollo utilizada generalmente en proyectos de corto plazo donde el equipo de trabajo y el tiempo para la entrega son pequeños. Está basada en una programación rápida o extrema, en la comunicación y el reciclado continuo de código, de lo que depende el éxito del proyecto. Un requisito fundamental para alcanzar el objetivo trazado es que el cliente debe ser parte del equipo de desarrollo brindando la retroalimentación necesaria. (9)

Esta metodología consta de seis fases:

1. Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración puede

tomar de pocas semanas a pocos meses dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología seleccionada. (10)

2. Planificación de la entrega

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Esta fase dura unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. (10)

3. Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración para maximizar el valor del negocio. Al final de la última iteración el sistema estará listo para entrar en producción. Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores. (10)

4. Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Es posible que se rebaje el tiempo que toma cada iteración, de tres semanas a una. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento). (10)

5. Mantenimiento

Mientras la primera versión se encuentra en producción, se debe mantener el sistema en funcionamiento al mismo tiempo que desarrollan nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente de manera tal que la velocidad de desarrollo pueda disminuir después de poner el sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura. (10)

6. Muerte del proyecto

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo. (10)

La metodología XP se basa en:

- *La re fabricación:* se crean patrones o estándares que permiten que sea posible la reutilización del código de manera más flexible.
- *Pruebas unitarias:* permite descubrir posibles errores futuros por medio de pruebas a los principales procesos.
- *Programación en pares:* dos desarrolladores participan en un proyecto en una misma estación de trabajo.

Entre sus ventajas se puede apreciar que:

- La programación se lleva a cabo de manera organizada.
- La taza de errores es menor.
- La satisfacción del programador.

Presenta desventajas como:

- Altos costos en caso de que el proyecto falle.

- Es recomendable llevarla a cabo con un equipo de programadores avanzados.
- Resulta difícil realizar mejoras o contratar personal nuevo en caso de que no se haga la documentación.

(11)

Para el desarrollo de la solución de software de esta investigación se escogió como metodología de desarrollo XP, principalmente por sus propiedades flexibles que aumentan la productividad y la resistencia al cambio de requerimientos, adaptándolo a las necesidades reales del desarrollo del marco de trabajo Kairos, donde toma el rol de cliente su equipo de desarrollo que son los usuarios finales de la herramienta. Además con su uso se podrán tener continuas versiones del producto y la agregación frecuente de nuevas funcionalidades.

1.7 Herramientas CASE

Las herramientas de Ingeniería de Software Asistida por Computadora (CASE por sus siglas en inglés) son software destinados a incrementar la productividad en el proceso de desarrollo de software, reduciendo los costos de tiempo y recursos. Entre sus principales características se encuentran las facilidades que brindan para la realización de prototipos y el desarrollo conjunto de aplicaciones, facilita la reutilización de componentes de software y mejora y estandariza la documentación. (12)

1.7.1 Visual Paradigm for UML v3.4

Es una herramienta de modelado visual para todo tipo de diagramas UML (Lenguaje Unificado de Modelado), desarrollada por la empresa Visual Paradigm Internacional. Esta herramienta está diseñada para el desarrollo de software orientado a objetos brindando soporte para todo su ciclo de vida, así como muchas facilidades para el trabajo colaborativo, además de la integración para el trabajo con modelos relacionales de Bases de Datos. (13) Permite la generación de código en diferentes lenguajes entre los que se encuentran Java, C# y Python. Ofrece la posibilidad de mantener el código generado y el modelo de diseño sincronizados a través de todo el ciclo de desarrollo, puede ejecutarse en varias plataformas como Linux y Windows y la universidad cuenta con la licencia para su explotación y.

1.7.2 IBM Rational Rose Enterprise Edition v7.0

Es una herramienta desarrollada originalmente por la empresa Rational Machines adquirida por IBM en el 2003, la cual propone la utilización de cuatro tipos de modelos para realizar el diseño del sistema, utilizando una vista estática y otra dinámica, que pueden ser creadas y refinadas realizando de esta forma un esquema completo que representa el dominio del problema y la solución de software. Rational Rose permite que los desarrolladores operen en un espacio de trabajo privado donde tienen el módulo completo, controlando exclusivamente la propagación de los cambios en el mismo, por lo que pueden trabajar a la vez varias personas en este proceso. A pesar de contar con características favorables presenta desventajas significativas ya que para su utilización requiere de una alta capacidad de procesamiento y de un alto costo para su adquisición. (14)

Como herramienta CASE se selecciona Visual Paradigm for UML en su versión 3.4 por ser de sencilla utilización, además la UCI posee la licencia que permite su empleo en proyectos productivos y permite la generación de código Java a partir de los Diagramas de clases del diseño.

1.8 Lenguaje de modelado

Los lenguajes de modelado son conjuntos de símbolos estándares que permiten crear diagramas de acuerdo a la forma en que son unidos y rigiéndose por reglas.

1.8.1 Lenguaje Unificado de Modelado v2.0

Es uno de los lenguajes de modelado más utilizado en la actualidad, está compuesto por varios elementos gráficos, los que combinados conforman diagramas, rigiéndose por un conjunto de reglas. Se utiliza para definir un sistema, para detallar los artefactos en el mismo, para documentar y construir. UML describe lo que supuestamente hará un sistema, pero no dice cómo implementarlo. Además proporciona gran variedad de formas para soportar una metodología, pero no se enmarca en ninguna en específico por lo que puede ser utilizado con XP, que es la metodología seleccionada para llevar a cabo esta investigación.

1.9 Entornos de Desarrollo Integrado

Los entornos de desarrollo integrado (IDE por sus siglas en inglés) son herramientas que ofrecen a los desarrolladores varias facilidades para llevar a cabo la realización de un software, como pueden ser la edición, compilación y depuración de código, la integración con gestores de Bases de Datos permitiendo realizar operaciones sobre ellas, con servidores de aplicaciones y controlador de versiones. Incluso hay algunos que permiten el modelado de clases de forma visual. (15)

Existen varios IDEs que podrían satisfaces las necesidades en cuanto al desarrollo del componente de seguridad para el marco de trabajo Kairos, pero se decidió hacer un análisis de los más utilizados en la UCI para el lenguaje Java, los cuales son NetBeans y Eclipse.

1.9. 1 NetBeans v7.0.1

Es un IDE libre y gratuito, que permite escribir, compilar, depurar y ejecutar aplicaciones de escritorio, web y empresariales. Posee buena integración con algunas de las herramientas que se utilizaran en el desarrollo de la solución, como son el servidor GlassFish Open Source Edition 3.0.1, el controlador de versiones Subvertion y el gestor de Base de Datos PostgreSQL 8.4. Brinda un excelente soporte para la plataforma JEE, incluyendo persistencia y EJB 3.0, además de un entorno para el diseño de aplicaciones gráficas muy amigable, en el que las interfaces de usuario se modelan de manera visual utilizando el marco de trabajo Swing.

1.9. 2 Eclipse Índigo v3.7

Es un IDE libre de código abierto, utiliza plugins para desempeñar sus funciones, por lo que es fácilmente extensible. De esta forma se puede adaptar para otros lenguajes de programación a parte de Java, que es para el que está configurado inicialmente, entre estos lenguajes se encuentran C, C++, y Phyton. Posee buena integración con el controlador de versiones CVS, así como varios gestores de Bases de Datos. Brinda soporte para el desarrollo de aplicaciones con el marco de trabajos Spring e Hibernate, con auto completamiento de la sintaxis para los ficheros de configuración XML y los nombres de clases para dichos ficheros. No permite el diseño de aplicaciones de escritorio de manera visual, sin embargo se le puede instalar un plugin para esto.

Aunque ambos IDEs poseen características que los hacen apropiados para desarrollar esta propuesta, se seleccionó NetBeans 7.0.1 ya que permite el desarrollo de todo tipo de aplicación en Java. Posee buena integración con algunas de las herramientas que se utilizaran en el desarrollo de la solución, como son el servidor de aplicaciones GlassFish Open Source Edition 3.0.1 y el gestor de Base de Datos PostgreSQL 8.4. Además mejora el soporte de versiones anteriores para el desarrollo de aplicaciones empresariales en Java.

1.10 Servidor de aplicaciones

Actualmente existen varios servidores contenedores de aplicaciones para la tecnología Java en el mercado con muy buenas prestaciones, características interesantes, un gran número de descargas en internet y una gran cantidad de productos software registrados que se ejecutan sobre ellos. De estos, para la UCI, CEGEL y por ende para el desarrollo de esta investigación el servidor GlassFish Open Source Edition 3.0.1 es una buena opción ya que es desarrollado y mantenido por una amplia comunidad de software, de código abierto, libre y gratuito. Fue uno de los primeros en implementar la plataforma JEE6. (16) Soporta clusterización y balanceo de carga, además de la especificación de Integración de Negocio de Java (JBI por sus siglas en inglés), por lo que proporciona un entorno unificado de instalación, administración y monitoreo, con un sustento excelente provisto por el IDE NetBeans 7.0.1 que posibilita desplegar fácilmente las aplicaciones que se desarrollen así como administrar los recursos del servidor desde la propia herramienta. El perfil de clústeres permite la replicación de sesiones en memoria, proporcionando un enfoque efectivo para mejorar la disponibilidad. Provee una infraestructura de administración amigable que puede ser controlada a través de una consola de administración web que tiene dicho servidor, esta facilidad es una característica muy rara en los software libres, por lo que el GlassFish Open Source Edition 3.0.1 le lleva en este sentido gran ventaja a sus semejantes de libre distribución. Brinda gran flexibilidad para realizar la autenticación de los usuarios por medio de los dominios de seguridad. (17)

1.10. 1 Dominios de seguridad en el GlassFish Open Source Edition v3.0.1

Los servidores de aplicaciones para la JEE necesitan interactuar con gran cantidad de sistemas externos para cumplir con los requerimientos de una determinada organización y el GlassFish Open Source Edition

3.0.1 no es la excepción. Entre estos sistemas externos se encuentran los Sistemas de almacenamiento de identidad, los cuales almacenan información de identificación como puede ser el nombre de usuario y credenciales como la contraseña.

Los dominios de seguridad proveen una forma estándar para los servidores de aplicaciones autenticar las credenciales provistas por el usuario o cualquier entidad de software o hardware contra un conjunto de datos de identificación en algún Sistema de almacenamiento de identidad.

El servidor GlassFish Open Source Edition 3.0.1 proporciona varios dominios de seguridad por defecto, entre los que se encuentran un dominio para Base de Datos, uno para servidor LDAP y otro para ficheros planos. A pesar de esto en ocasiones es necesario realizar la autenticación contra un Sistema de almacenamiento de identidad no soportado por estos. (17) En ese caso se debe implementar un dominio de seguridad propio y configurarlo para que sea utilizado por el servidor GlassFish a la hora de autenticar un usuario determinado lo cual forma parte del mecanismo de autenticación desarrollado en la presente investigación y será tratado en el próximo capítulo.

Otra de las tecnologías que proporciona el servidor GlassFish Open Source Edition 3.0.1 es una implementación de la especificación EJB 3.0.

1.10. 2 EJB v3.0

EJB es la tecnología del lado del servidor para la arquitectura de componentes de la plataforma JEE que permite un desarrollo rápido y simplificado de aplicaciones distribuidas, transaccionales, seguras y portables. Es una de las API que forman parte del estándar de construcción de aplicaciones empresariales y su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor.

El contenedor EJB proporcionado por el servidor GlassFish Open Source Edition 3.0.1 proporciona varios servicios añadidos, entre los que se encuentran: (18)

- **Seguridad:** comprobación de permisos de acceso a los métodos de los beans empresariales.
- **Concurrencia:** llamada simultánea a un mismo bean empresarial desde múltiples clientes.
- **Servicios de red:** comunicación entre el cliente y el bean empresarial en máquinas distintas.

- **Persistencia:** sincronización entre los datos de las clases persistentes y tablas de una Base de Datos.
- **Escalabilidad:** posibilidad de constituir clústeres de servidores de aplicaciones con múltiples nodos para poder dar respuesta a aumentos repentinos de carga de la aplicación con sólo añadir algunos de ellos.
- **Adaptación en tiempo de despliegue:** posibilidad de modificación de todas estas características en el momento del despliegue del bean.

1.11 Sistemas Gestores de Bases de Datos

Los Sistemas Gestores de Base de Datos (SGBD) son sistemas cuyo objetivo es el de servir de interfaz entre la Base de Datos, los usuarios y las aplicaciones, permitiendo a los desarrolladores abstraerse de la forma en que es almacenada la información. Están compuestos generalmente por un lenguaje de definición, manipulación y consulta de datos y deben asegurar la consistencia, integridad, seguridad e independencia de la información que manejan, así como brindar un mecanismo de respaldo y recuperación eficiente. (19)

1.12. 1 PostgreSQL v8.4

Es un SGBD de código abierto desarrollado originalmente en la Universidad de California en Berkeley. Brinda conceptos básicos adicionales que facilitan la extensión de los sistemas, como son la herencia, las clases, los tipos de datos y las funciones. Estas características lo colocan en la categoría de las Bases de Datos que se identifican como objeto-relacionales, cuenta además con las restricciones (constrains), los disparadores (triggers), las reglas (rules) y la integridad transaccional.

Es multiplataforma compatible con varios de los sistemas operativos actuales, así como el estándar ANSI-SQL 92/99. Puede manejar gran cantidad de conexiones de forma concurrente, a través de un sistema llamado Acceso Concurrente Multiversión (20), lo que posibilita que mientras un proceso realiza una escritura en una tabla otros puedan acceder a esta sin necesidad de bloqueos. Puede integrarse a un sistema distribuido compuesto por una variedad de recursos, como pueden ser otra Base de Datos en Oracle y una cola de mensajes que al igual soporta funciones escritas en un buen número de lenguajes

como C, C++, Java PL, PL/PHP y PL/Python permitiendo a los desarrolladores sacar partido de las ventajas de cada uno de estos lenguajes. (21)

1.12. 2 Oracle DB Edición Empresarial

Es un SGBD propietario desarrollado por la Corporación Oracle, entre las características que más se destacan se encuentran su estabilidad, el soporte de transacciones, la escalabilidad y la capacidad de ejecución en varias plataformas. Soporta los disparadores (triggers), las reglas (rules) y los procedimientos almacenados.

Además viene integrado con la Oracle Real Application Clusters (Oracle RAC), lo cual permite a una sola Base de Datos ejecutarse en un clúster de servidores proporcionando una tolerancia a fallos inmejorable, un rendimiento y escalabilidad sin necesidad de cambios en las aplicaciones clientes. El rendimiento de la Base de Datos es incrementado mediante la compresión de los datos en particiones de almacenamiento de bajo costo.

Ofrece un potente desempeño de Bases de Datos con bloqueo, control de acceso con múltiples factores y privilegios de usuario, clasificación de datos, encriptación de datos transparente, auditoría e informes consolidados, administración de configuración segura y enmascaramiento de datos para que los clientes puedan implementar soluciones de seguridad de datos fiables sin necesidad de modificar sus aplicaciones, soporta el procesamiento analítico en línea y la minería de datos. El costo de una licencia ronda los 47000 dólares. (22)

Luego de haber analizado las características de ambos gestores se seleccionó PostgreSQL en su versión 8.4 teniendo en cuenta principalmente su licencia gratuita, que es multiplataforma y es además el gestor que utiliza el marco de trabajo Kairos.

1.12 Pruebas de unidad

Una de las fases más importantes del ciclo de vida de un software es la fase de pruebas. Se estima que la mitad del esfuerzo de desarrollo de un programa tanto en tiempo como en gastos se consume en esta fase, de ahí la importancia de ésta. Las pruebas son de suma importancia para el software y tienen sus implicaciones en la calidad de éste, citando a Deutsch: “El desarrollo de sistemas de software implica una serie de actividades de producción en las que las posibilidades de que aparezca el fallo humano son

enormes. Debido a la imposibilidad humana de trabajar y comunicarse de forma perfecta, el desarrollo de software ha de ir acompañado de una actividad que garantice la calidad". (23) Uno de los pasos para garantizar que el producto final tenga la menor cantidad de errores posibles es el desarrollo de un correcto procedimiento de pruebas de unidad.

Para realizar este tipo de pruebas a la solución de software desarrollada se utilizará el marco de trabajo JUnit en su versión 4.5, el cual es de código libre, gratuito y permite la ejecución de clases Java de manera controlada para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Además se encuentra integrado al IDE NetBeans 7.0.1 que permite la generación de las clases de pruebas y visualizar los resultados de las pruebas en forma de árbol o de reporte con formato html.

1.13 Conclusiones del capítulo

En el presente capítulo se realizó una investigación sobre diferentes elementos teóricos relacionados con la seguridad de aplicaciones como la integridad y confidencialidad de la información, la autenticación y la autorización. Se realizó un análisis de las principales herramientas, metodologías y tecnologías necesarias para llevar a cabo el desarrollo de los mecanismos de seguridad requeridos en el marco de trabajo Kairos.

De esta forma los algoritmos de resumen con que contará esta solución para la encriptación de las contraseñas de las cuentas de usuario y la verificación de la integridad de la información serán md5 y sha-1 y el algoritmo de clave simétrica Triple-DES para el cifrado de la información. Se utilizará NetBeans en su versión 7.0.1 como IDE de desarrollo por ser uno de los más completos para Java además de ser libre y poseer buena integración con el servidor de aplicaciones GlassFish Open Source Edition 3.0.1. Como metodología de desarrollo se selecciona XP que posibilita un desarrollo ágil de la solución. Como herramienta CASE se utilizará Visual Paradigm en su versión 3.4 utilizándose el lenguaje de modelado de sistemas UML en su versión 2.0 para la generación de algunos diagramas. Las pruebas de unidad se realizarán utilizando el marco de trabajo JUnit 4.5 ya que posee muy buena integración con el IDE seleccionado.

Capítulo 2: Descripción y diseño de los mecanismos de la propuesta de solución

2.1 Introducción

En este capítulo se presenta el diseño de la solución de software desarrollada durante esta investigación así como la descripción de los mecanismos de seguridad definidos, además se exponen los estilos y patrones arquitectónicos utilizados al efecto. Como resultado se obtuvo el diagrama de clases del mecanismo desarrollado.

2.2 Propuesta de solución

Después de haber realizado un análisis de las deficiencias de seguridad existentes en el marco de trabajo Kairos, su equipo de desarrollo propone definir y desarrollar un mecanismo que garantice la confidencialidad e integridad de la información que se intercambia a través de la red entre los clientes y el servidor de aplicaciones, facilitando además la autenticación de los clientes. Otra de las funcionalidades de importancia con que debe contar es el control de las fortalezas de las contraseñas de las cuentas de usuarios, lo que posibilitará disminuir el riesgo ante ataques de fuerza bruta, también es importante controlar el tiempo de inactividad máximo permitido sin que un operador interactúe con la aplicación mitigando el riesgo de un acceso no autorizado a la misma. La encriptación de las contraseñas de las cuentas de usuario es otro de los mecanismos de importancia para evitar el robo de estas lo que comprometería toda la seguridad del sistema.

Es necesario que este mecanismo de seguridad sea fácil de utilizar y configurar, por lo cual es necesario proporcionar a los clientes un nivel de abstracción adecuado de las complejidades de mecanismos tales como el de confidencialidad e integridad.

Una vez que se hayan implementado estas funcionalidades y sean acopladas al marco de trabajo Kairos se espera que esta solución contribuya de manera importante al aseguramiento de la información manejada por las aplicaciones que lo utilicen.

2.3 Diseño de la solución

2.3.1 Patrones GRASP

Los Patrones Generales de Software para la Asignación de Responsabilidades (GRASP por sus siglas en inglés) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

Experto: Este patrón permite asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. La responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantendrá encapsulada (disminución del acoplamiento). Se pone en práctica en los gestores de solicitud, los manejadores de envío y recepción así como en los empaquetadores y desempaquetadores.

Ventajas:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. (23)
- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase “sencillas” y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión. (23)

Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que asuma la creación de un objeto en un momento determinado.

Ventajas: (23)

- Brinda soporte a un bajo acoplamiento, lo cual supone poca dependencia respecto al mantenimiento y mejores oportunidades de reutilización.

Bajo Acoplamiento: Su principal característica es mantener las clases más independientes entre sí y con la menor cantidad de relaciones; la cual posibilita que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y

disminuyendo la dependencia entre las clases, asignándoles una responsabilidad para mantener un bajo acoplamiento.

Ventajas: (23)

- No se afectan por cambios de otros componentes.
- Fácil de entender por separado.
- Fácil de reutilizar.

Alta Cohesión: La principal característica de este patrón es asignar responsabilidades de modo que la cohesión siga siendo alta. La información que almacena una clase debe de ser coherente y debe estar en la medida de lo posible relacionada con la clase.

Ventajas: (23)

- Mejoran la claridad y la facilidad del diseño.
- Simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo, se genera un bajo acoplamiento.
- Soporta una mayor capacidad de reutilización.

2.3.2 Patrones GoF

Los patrones GoF (Gang-Of-Four) son soluciones a problemas comunes en el desarrollo de software. Durante el desarrollo de esta solución fue necesaria la aplicación de algunos de estos patrones los cuales permiten la reutilización de código y reducir el impacto de los cambios de manera considerable.

Patrones creacionales

- **Método factoría (Factory Method):** Centraliza en un método la creación de objetos de un subtipo determinado, ocultando al usuario la casuística para realizar la selección. Este patrón se utiliza en la solución de software de esta investigación en la creación de la estrategia de generación de la llave de cifrado a partir de una cadena de texto compuesta por los identificadores de los algoritmos de generación requeridos, además se utiliza en la creación de los objetos encargados de generar el resumen de la información y de cifrar la misma de acuerdo a un algoritmo determinado.

- **Instancia única (Singleton):** Garantiza la existencia de una única instancia para una clase durante la vida de la aplicación y la creación de un mecanismo de acceso global a dicha instancia. Este patrón permitirá la existencia de solo una instancia de la clase que contiene la configuración de los mecanismos de esta solución así como una vía sencilla para acceder a esta.

Patrones estructurales

- **Proxy:** Actúa como intermediario entre un objeto cliente y el objeto receptor. En esta solución es decisivo en el desarrollo del mecanismo de abstracción permitiendo ocultar a los clientes los procesos de cifrado y verificación de integridad de la información para lograr una comunicación segura.
- **Envoltorio (Decorator):** Añade funcionalidad a una clase dinámicamente sin necesidad de utilizar la herencia lo cual propicia que la cantidad de clases en una jerarquía se mantenga bajo, además permite adicionar funcionalidad a un objeto en tiempo de ejecución sin modificar la estructura del mismo. Es utilizado en la solución de software de esta investigación para extender la funcionalidad de las clases encargadas del cálculo de resúmenes de información a la hora de encriptar una contraseña, permitiendo duplicar la longitud de los resúmenes.

Patrones de comportamiento

- **Estrategia (Strategy):** Es útil cuando existen varios algoritmos que un cliente debe seleccionar dinámicamente. Este patrón sugiere mantener la implementación de cada algoritmo en una clase separada e implementando una interfaz común. En la solución de software de esta investigación es aplicado en las clases de estrategia de generación de la llave de cifrado, donde para cada algoritmo existe una clase, además se aplica a las clases encargadas del cálculo de resúmenes donde para cada algoritmo de resumen existe una clase, implementando siempre en ambos casos una interfaz común.
- **Objeto Nulo:** Propone la creación de un objeto en una jerarquía de clases que no contiene ninguna funcionalidad sino que su objetivo es evitar a los clientes la verificación de nulidad. De esta forma si en la jerarquía no se encuentra disponible ningún objeto se devuelve al cliente un objeto nulo. Es utilizado en esta solución en la jerarquía de estrategias para la generación de la llave de cifrado.

2.3.3 Tarjetas CRC

Para poder diseñar el sistema como un equipo se debe cumplir con tres principios: Cargo o Clase, Responsabilidad y Colaboración, los que se logran por medio de la utilización de las tarjetas CRC que permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos, además permiten que todo el equipo contribuya en la tarea del diseño.

A continuación aparecen algunas de las tarjetas CRC creadas durante el diseño de esta solución, específicamente las tarjetas mostradas están relacionadas con el proceso de encapsulamiento de la información que se transmite entre cliente y servidor así como el cifrado y resumen de la misma que son algunas de las operaciones críticas en la solución de software de esta investigación. El resto de las tarjetas se pueden consultar en el Anexo3.

Plantilla para las tarjetas CRC

Tarjeta CRC	
Clase: Nombre de la clase que se está modelando.	
Súper clase: Nombre de la clase padre en la herencia.	
Sub clase: Nombres de las clases hijas en la herencia.	
Responsabilidades: Es una descripción de alto nivel de las funcionalidades de la clase.	**Colaborador:** Indica con cuáles otras clases se requiere colaborar para cumplir la responsabilidad.

Tarjeta CRC EmpaquetadorConfidencialidad

Tarjeta CRC
Clase: EmpaquetadorConfidencialidad

Súper clase: Empaquetador	
Sub clase: -	
Responsabilidades: Cifrar la información que se va a transmitir encapsulándola en un paquete.	**Colaborador:** Cifrador, Paquete

Tarjeta CRC EmpaquetadorIntegridad

Tarjeta CRC	
Clase: EmpaquetadorIntegridad	
Súper clase: Empaquetador	
Sub clase: -	
Responsabilidades: Calcular un resumen de la información que se va a transmitir y encapsular la información junto al resumen.	**Colaborador:** Resumen, Paquete

Tarjeta CRC EmpaquetadorInvocacion

Tarjeta CRC	
Clase: EmpaquetadorInvocacion	
Súper clase: Empaquetador	
Sub clase: -	
Responsabilidades: Encapsular la información necesaria para invocar dinámicamente una	**Colaborador:**

función en un objeto remoto.	Paquete

Tarjeta CRC Cifrador3DES

Tarjeta CRC	
Clase: Cifrador3DES	
Súper clase: Cifrador	
Sub clase: -	
Responsabilidades: Cifrar un conjunto de bytes utilizando el algoritmo Triple-DES.	**Colaborador:** UtilComun

Tarjeta CRC ResumenMd5

Tarjeta CRC	
Clase: ResumenMd5	
Súper clase: Resumen	
Sub clase: -	
Responsabilidades: Calcular un resumen de un conjunto de bytes utilizando la función md5.	**Colaborador:** -

Tarjeta CRC ResumenSha1

Tarjeta CRC	
Clase: ResumenSha1	
Súper clase: Resumen	
Sub clase: -	
Responsabilidades: Calcular un resumen de un conjunto de bytes utilizando la función sha1.	**Colaborador:** -

2.3.4 Arquitectura de la solución

Debido a que el diseño arquitectónico del marco de trabajo Kairos está enfocado en un estilo Cliente-Servidor, la solución de software de la presente investigación se basa en el mismo estilo, el cual permitirá contar con una solución distribuida donde se podrá establecer una comunicación contractual entre funcionalidades que se ejecutan en las estaciones clientes y otras en el servidor para dar solución a los requerimientos de los usuarios, abstrayéndolos de la distribución física de la solución.

A pesar de que el estilo arquitectónico Cliente-Servidor proporciona un alto nivel organizativo, no es suficiente ya que el diseño de estas aplicaciones continúa siendo complejo. Para organizar mejor el diseño de estas partes se utilizará el estilo n-capas teniendo en cuenta que este permite establecer una organización jerárquica entre cada una de las capas, las cuales agrupan funcionalidades similares, garantizando que cada capa proporcione sus servicios a la capa inmediata superior y que esta a su vez se sirva de las prestaciones que le brinda la capa inmediata inferior. (24)

Este diseño arquitectónico se orienta a la reutilización de componentes, con el fin de reducir el ciclo de desarrollo y proporcionar un diseño más escalable y refinado. El desarrollo basado en componentes permitirá reutilizar componentes desarrollados previamente sin tener en cuenta la composición de los mismos, sumándole un incremento a la mantenibilidad considerable, debido a que la actualización de estos será transparente para el sistema, siempre y cuando no se modifique la interfaz de interacción con

el resto del sistema. A continuación se muestra una imagen con la estructura de capas con que cuenta la solución de software. *Ver Fig. 3.*

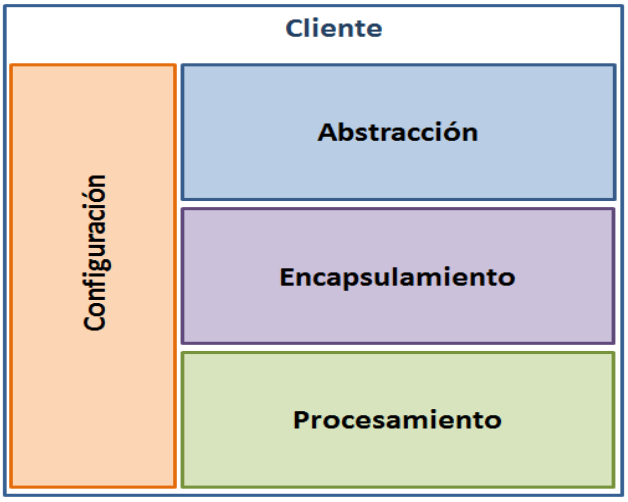


Fig. 3 Arquitectura de la solución de software.

La parte cliente cuenta con cuatro capas, tres de ellas horizontales y una vertical. La capa de configuración contiene la lógica encargada de cargar las configuraciones que provee el cliente e inicializar diferentes objetos que luego son necesitados por otras capas. La capa de abstracción se encarga de interactuar con los objetos clientes proporcionando las interfaces para acceder a las funcionalidades de la solución, además cuenta con la lógica para garantizar la abstracción de las complejidades de los procesos de confidencialidad y verificación de la integridad de la información. Esta capa se apoya en la capa de encapsulamiento, la cual es la encargada de empaquetar la información siguiendo un modelo de paquetes bien definido lo que permite una correcta comunicación. La capa de procesamiento brinda sus servicios a la capa de encapsulamiento, en la misma se encuentran funcionalidades como el cálculo de resumen y encriptación de la contraseña así como el de cifrado y descifrado de información.

La parte de esta propuesta que se ejecuta en el servidor cuenta con las mismas capas ya que presenta los mismos procesos que el cliente.

2.3.5 Diagramas de clases

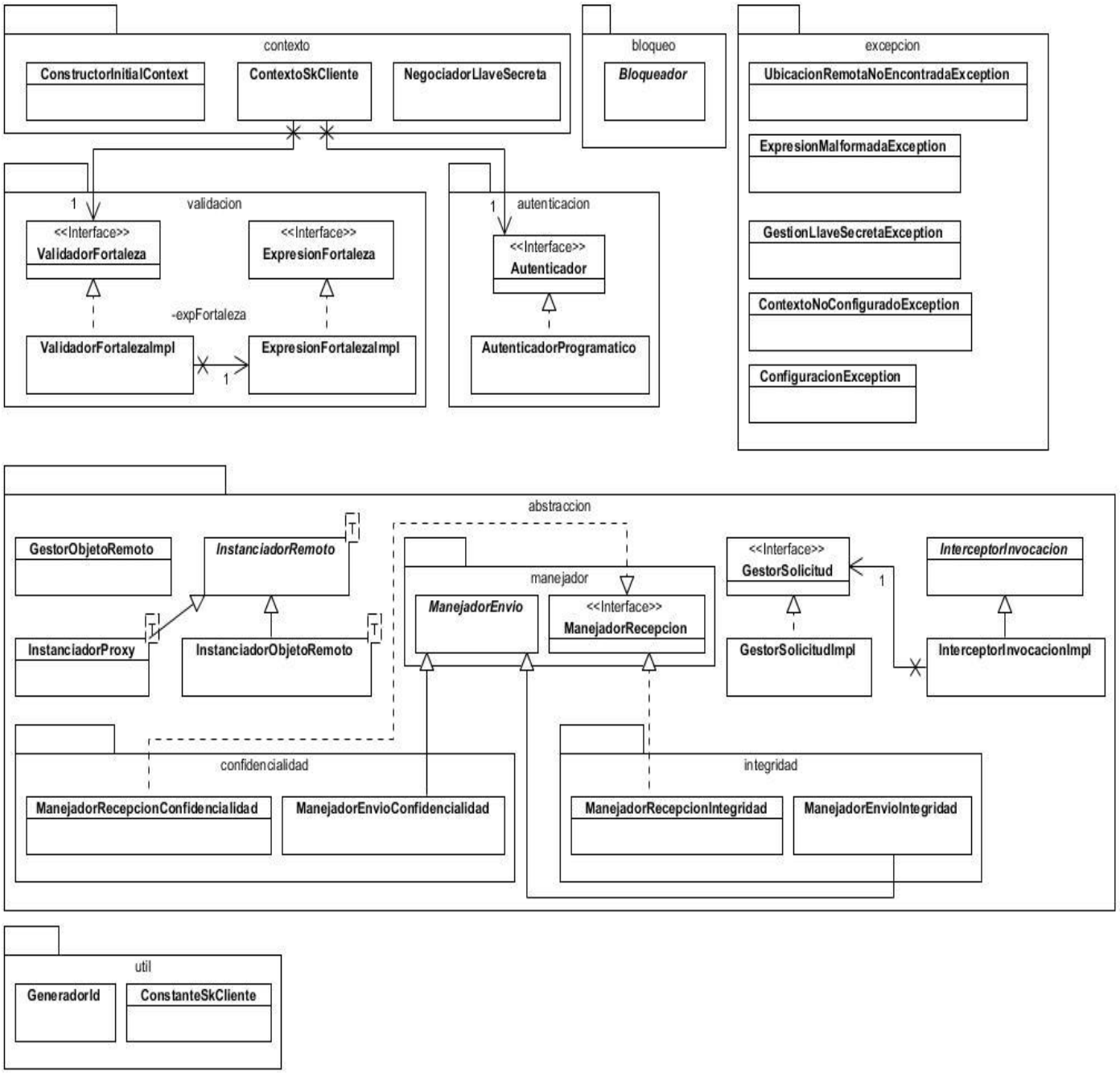


Fig. 4 Diagrama de clases de la librería sk-cliente.jar.

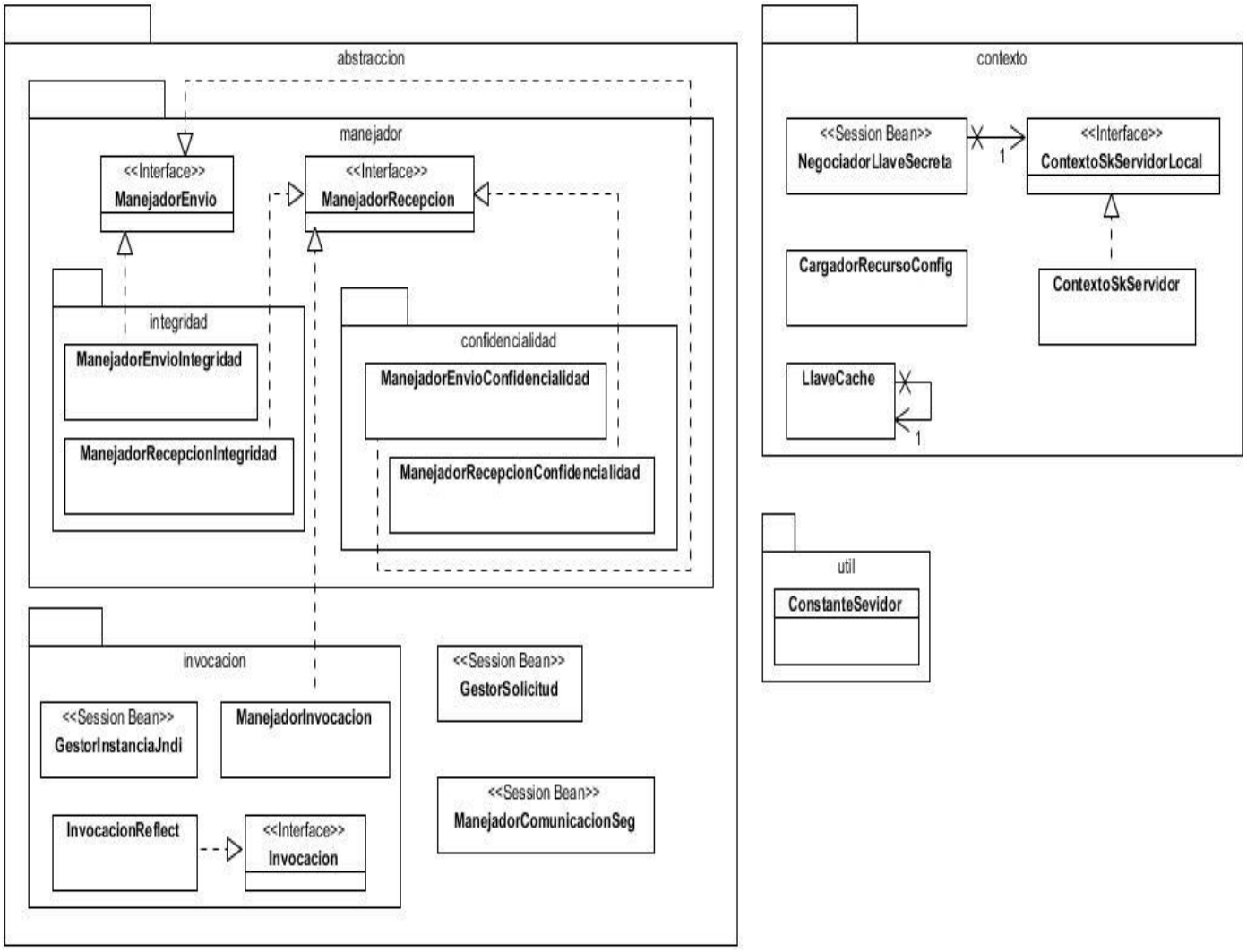


Fig. 5 Diagrama de clases de la librería sk-servidor.jar.

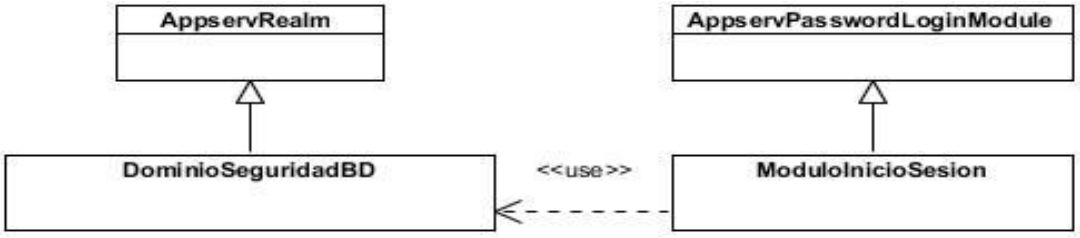


Fig. 6 Diagrama de clases de la librería dominio-seg-jdbc.jar.

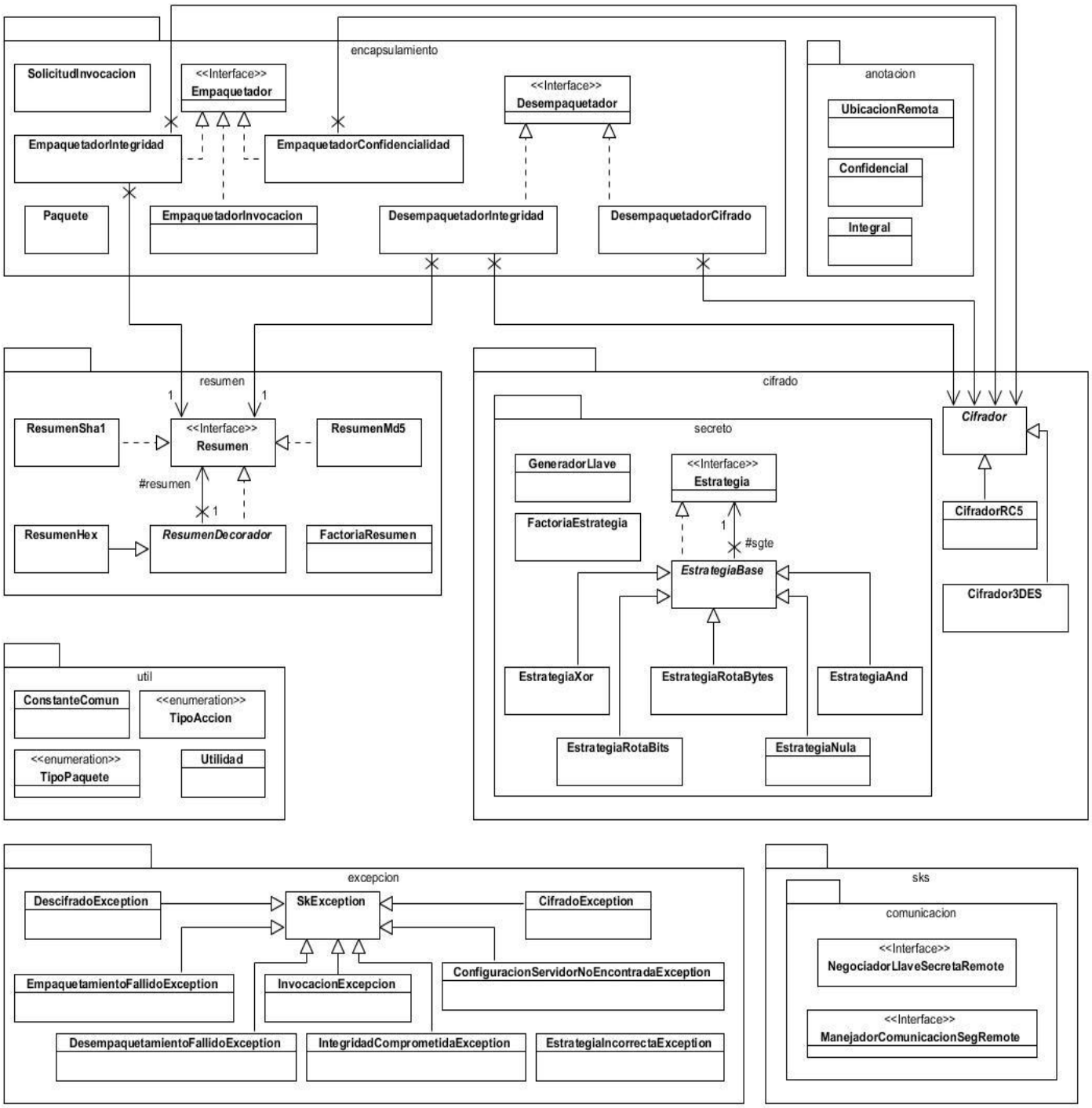


Fig. 7 Diagrama de clases de la librería sk-comun.jar.

2.4 Definición de los mecanismos de seguridad

La parte fundamental de esta investigación se centra en la definición de un conjunto de mecanismos de seguridad que permitan al marco de trabajo Kairos el aseguramiento de la información que manejan las aplicaciones que lo utilicen.

2.4.1 Mecanismo de autenticación y autorización

Con vista a autenticar los usuarios contra el servidor de aplicaciones GlassFish Open Source Edition 3.0.1 y definir la vía para controlar el acceso a las funcionalidades publicadas en este fue necesario definir y desarrollar el mecanismo de autenticación y autorización. Una parte fundamental del mismo es la implementación de un dominio de seguridad, el cual actúa como intermediario entre el servidor de aplicaciones y la Base de Datos donde se encuentra la información de identificación de las cuentas de usuarios, además se implementaron las funcionalidades requeridas para llevar a cabo la autenticación desde una aplicación cliente y se define cómo se debe realizar la autorización en el servidor de aplicaciones.

2.4.1.1 Dominio de Seguridad

Los dominios de seguridad permiten al servidor interactuar con distintos sistemas de almacenamiento de identidad para llevar a cabo la autenticación de los usuarios. Como parte del mecanismo de autenticación desarrollado en esta investigación se hizo necesario implementar un dominio de seguridad que permitiera autenticar los usuarios contra una Base de Datos.

Para implementar el mismo el servidor GlassFish Open Source Edition 3.0.1 proporciona dos clases que se deben extender, la clase *com.sun.appserv.security.AppservPasswordLoginModule* y *com.sun.appserv.security.AppservRealm* las cuales se encuentran en la librería **security.jar** que forma parte de las librerías del servidor GlassFish Open Source Edition 3.0.1.

La primera de estas clases proporciona la base para la implementación del módulo de inicio de sesión con el que el servidor interactúa directamente a la hora de autenticar un usuario. Para esto se debe implementar el método *protected abstract void authenticateUser() throws LoginException.* (17)

La segunda proporciona la base para la creación del dominio de seguridad propiamente y es la encargada de interactuar con el sistema de almacenamiento de identidad que en este caso es una Base de Datos. El principal método que se debe implementar de esta clase es *public void init(Properties props) throws BadRealmException, NoSuchRealmException*, el cual recibe como argumento las propiedades de configuración de este dominio y se encarga de inicializarlo. (17)

2.4.1.1.1 Configuración

El servidor GlassFish Open Source Edition 3.0.1 permite la configuración del dominio de seguridad de manera sencilla a través de su consola de administración web. Para ejecutar la misma se escribe en el navegador la dirección http://localhost:4848/ en caso de que el servidor se ejecute localmente. En otro caso se utiliza http://<host>:<puerto>. Otra variante es a través del IDE NetBeans 7.0.1, en la pestaña Servicios se despliega la opción “Servidores” y luego haciendo clic derecho sobre el icono del servidor seleccionamos la opción “Ver Consola de Administración”. Para mejor información consultar el anexo1.

Una vez que se encuentra abierta la consola se llega a la pantalla de configuración de los dominios de seguridad siguiendo la ruta Configuraciones/server-config/Seguridad/Dominios y luego se selecciona la opción “Nuevo”. Una vez que se realizan estos pasos se llega a la pantalla que se muestra en la *Fig. 8*. A continuación se llena el campo “Nombre” en este caso con *DominioSegBD* aunque podría ser cualquier nombre, luego se especifica la clase que representa el dominio de seguridad, para esto se selecciona la opción inferior en el campo “Nombre de la clase” y se escribe *seg.dominio.DiminioSegBD* que es el nombre de la clase que representa el dominio de seguridad desarrollado para esta solución. El siguiente paso y final es adicionar las propiedades de configuración necesarias para el funcionamiento del dominio. En el anexo2 se muestran las propiedades, su valor y una breve descripción de cada una de ellas.

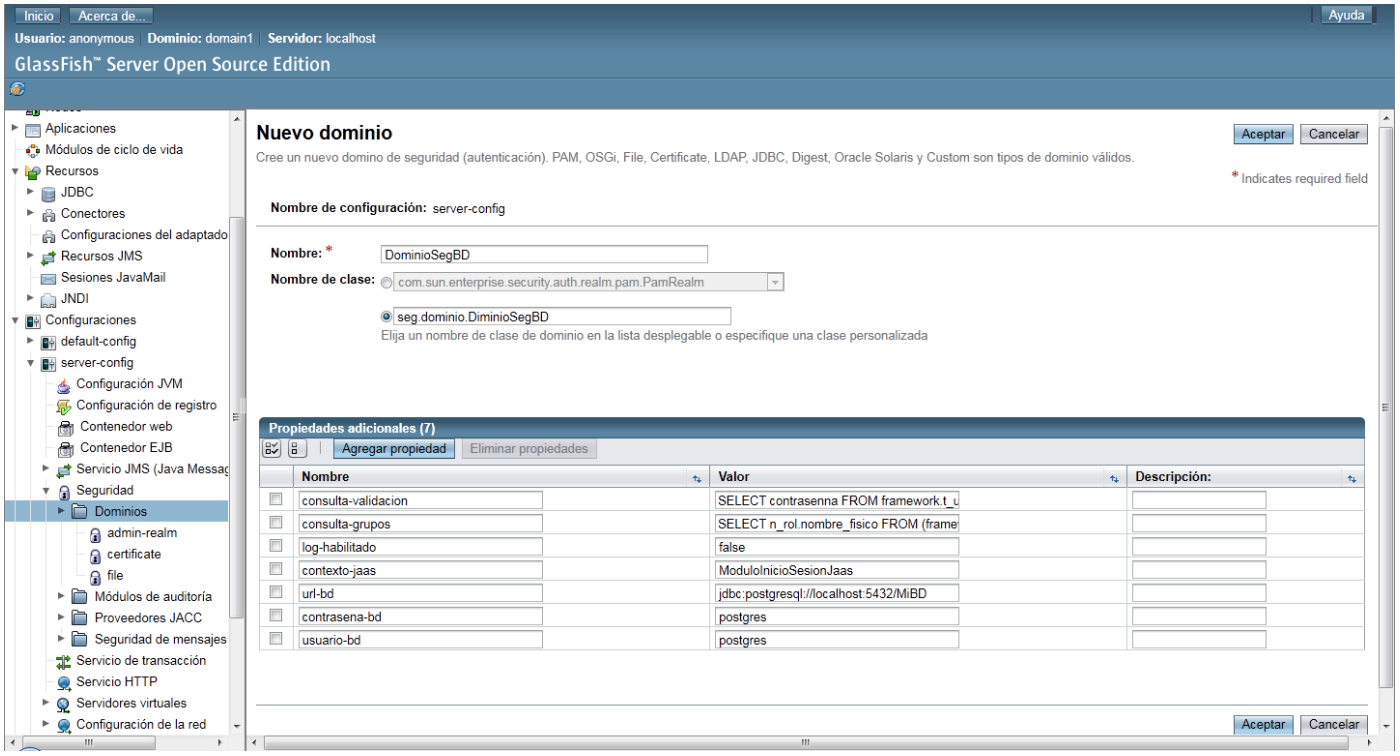


Fig. 8 Pantalla de configuración de los Dominios de Seguridad.

El siguiente paso es la configuración del módulo de inicio de sesión para que sea reconocido por el servidor. Esto se debe realizar de forma manual editando el fichero **login.conf** que se encuentra en <dir instalación servidor>/glassfish/domains/<dominio>/config/. Donde el elemento <dominio> es el nombre del dominio activo en el servidor, el mismo se puede observar en la parte superior de la consola de administración (en este caso es domain1).

En este fichero se debe agregar en cualquier ubicación el módulo de inicio de sesión que se utilizará con el dominio de seguridad configurado anteriormente de la siguiente manera:

ModuloInicioSesionJAAS{

    seg.sesion.ModuloInicioSesionJaas required;

};

Donde el identificador (en negrita) es precisamente el valor de la propiedad de configuración **contexto-jaas** pasada al dominio de seguridad.

2.4.1.2 Autenticación desde el cliente

Para llevar a cabo el proceso de autenticación desde una aplicación cliente contra el servidor GlassFish Open Source Edition 3.0.1 se debe hacer uso de la clase *com.sun.appserv.security.ProgrammaticLogin*, la cual contiene la lógica encargada de publicar en el mismo la identidad del cliente para su posterior acceso.

Esta clase se encuentra en la librería **security.jar** provista por el mismo servidor y proporciona los métodos *void login(String usuario, String contraseña, String nombreDominio, boolean propagarError)* y *boolean logout()* que se encargan de iniciar una sesión y finalizarla respectivamente.

Es importante que este proceso se realice antes de que se trate de acceder a una funcionalidad en un objeto remoto que requiera control de acceso, puesto que de lo contrario se lanzaría una excepción del tipo EJBException.

2.4.1.3 Autorización en el servidor

Para verificar que quien intente invocar una funcionalidad determinada en el servidor cuenta con los permisos necesarios para hacerlo se realiza el proceso de autorización. El mismo es gestionado por el servidor de aplicaciones y está basado en el uso de roles. Un rol puede ser visto como un privilegio que posee un cliente para invocar cierta funcionalidad o acceder a cierto recurso en un ambiente asegurado.

Para llevar a cabo este proceso la plataforma Java Edición Empresarial soportada por el servidor GlassFish Open Source Edition 3.0.1 provee la anotación @RolesAllowed, la cual admite como argumento un conjunto de roles en forma de cadenas de texto que representan los roles con que debe contar un usuario para acceder al recurso anotado, y puede ser utilizada a nivel de método o a nivel de clase como se puede apreciar en la *Fig.9.*

```
@Stateless
@RolesAllowed({"administrador","jefearea"})
public class GestorNSSesion implements GestorNSSesionRemote {

   @RolesAllowed("administrador")
   public void adicionarSesionActiva(EDSesion sesion) throws Exception{...}

   ...
}
```

Fig.9 Ejemplo del uso de la anotación @RollesAllowed

Solamente se permite invocar el método *adicionarSesionActiva()* a los usuarios que posean entre sus roles asignados el de **administrador**, al resto de los métodos de esta clase podrán acceder los usuarios que posean los roles de **administrador** y **jefearea**.

Además de adicionar estas anotaciones en el código que se desea asegurar es necesario declarar estos roles en el fichero de despliegue de la aplicación empresarial **glassfish-application.xml**. En este fichero se asignan los roles por grupos de usuarios de la siguiente forma:

```
<glassfish-application>
    <security-role-mapping>
        <role-name>admin</role-name>
        <principal-name class-name="frm.seg.KairosPrincipal">principal</principal-name>
        <group-name>administradores</group-name>
    </security-role-mapping>
    <security-role-mapping>
    …
    </security-role-mapping>
</glassfish-application>
```

Donde <role-name> contiene el rol que se asigna al grupo de usuarios definido en <group-name>. El elemento <principal-name> es utilizado para representar una entidad, como un individuo, una corporación o quien sea que intente acceder a un recurso asegurado y un id de sesión.

2.4.2 Validación de la fortaleza de la contraseña

La validación de la fortaleza de las contraseñas es importante para prevenir el uso por parte de los usuarios de contraseñas débiles que propicien que un individuo sin autorización pueda entrar al sistema por medio de ataques como los de fuerza bruta. Por esta razón en la presente investigación se desarrolló un mecanismo que permite la configuración y el control de la fortaleza de las contraseñas de las cuentas de usuario.

El nivel de fortaleza de las contraseñas viene dado por la medición de parámetros como la cantidad total de caracteres, la cantidad de caracteres especiales, la cantidad total de dígitos y la cantidad de letras en mayúscula. Para establecer la cantidad mínima requerida de cada uno de estos se definió la expresión **ctc(+n);cce(+n);clm(+n);ctd(+n)**. En esta cada elemento separado por un punto y coma representa un parámetro que acepta un atributo indicando la cantidad requerida de apariciones en la contraseña. En la tabla que se muestra a continuación se describen los parámetros que componen esta expresión.

Tabla 1: Parámetros que componen la expresión de la fortaleza de la contraseña.

Parámetro	Descripción
ctc	Representa la cantidad total de caracteres.
cce	Representa la cantidad de caracteres especiales.
clm	Representa la cantidad de letras mayúsculas.
ctd	Representa la cantidad total de dígitos.

Si un atributo es un signo "*****" indica que se admiten cualquier cantidad de apariciones, si es "**+d**" (donde "**d**" es un número entero) indica que se requieren al menos "**d**" apariciones.

Para verificar la fortaleza de una contraseña en el marco de trabajo Kairos se debe crear un objeto de la interfaz *ValidadorFortaleza*, luego se invoca al método *void crearExpresion(String exp)* que se encarga de configurar el validador de acuerdo a la expresión de fortaleza recibida como argumento. Una vez hecho esto basta con invocar al método *boolean esValidaContrasena(char[] contra)* pasando como argumento la contraseña que desea validar.

2.4.3 Encriptación de contraseña

Uno de los rasgos de seguridad más importantes usados hoy en día en los sistemas de software son las contraseñas. Sin embargo, si alguna de las contraseñas del sistema cae en manos equivocadas pone en riesgo parte y, en el peor de los casos, toda la información gestionada. Por este motivo es necesario que las mismas sean almacenadas y transmitidas por la red o cualquier otro canal inseguro encriptadas evitando que una persona no autorizada pueda apoderarse de estas y acceder al sistema. La encriptación se propone llevar a cabo utilizando las funciones de resumen md5 y sha-1, las cuales son sólo de ida o irreversibles. Gracias a esto no es posible revertir la encriptación para obtener la contraseña a partir de su valor encriptado. Para cada una de estas funciones se propone desarrollar una clase encargada de realizar la encriptación por el algoritmo correspondiente. Una clase factoría se encarga de la creación de la representación del objeto que realiza la encriptación por el algoritmo requerido mediante una cadena que lo identifica y que será igual al nombre del algoritmo deseado.

2.4.4 Bloqueo de sesión por inactividad

En ocasiones las aplicaciones informáticas son abandonadas en la estación de trabajo y continúan ejecutándose sin la presencia de su operador, presentándose de esta forma la oportunidad de que una persona no autorizada haga uso de la aplicación poniendo en riesgo toda la seguridad del sistema y por ende de la información. Para mitigar este riesgo se propone el desarrollo de un mecanismo que permita bloquear la aplicación luego de un tiempo definido sin la actividad de un usuario.

Para satisfacer este requerimiento en la presente investigación se desarrolla la clase *Bloqueador* que cuenta con los mecanismos para detectar los eventos de teclado y ratón que se presenten en la

aplicación, de esta forma se controla el tiempo de inactividad, el cual consiste en el tiempo en que no ocurren dichos eventos. En caso de alcanzar el tiempo de inactividad máximo programado se invoca el método *abstract void bloquear()* el cual debe ser implementado para realizar las operaciones necesarias para bloquear la aplicación. Por esto en el marco de trabajo Kairos se propone la creación de la clase *ControladorInactividad* la cual debe extender la clase *Bloqueador* e implementar el método *abstract void bloquear()* donde se deben realizar operaciones como ocultar la interfaz principal y mostrar un cuadro de diálogo que permita la autenticación de un usuario para desbloquear la aplicación.

2.4.5 Mecanismo de confidencialidad e integridad

La información que se transmite por la red corre el riesgo de ser modificada durante su trayecto o ser interceptada por personas malintencionadas con los inconvenientes que de ello se derivan. Es por esto que en la presente investigación se definen los mecanismos necesarios para mitigar estos problemas.

2.4.5.1 Mecanismo de confidencialidad

El mecanismo de confidencialidad debe garantizar la transmisión de la información de forma tal que, de ser interceptada, no sea posible o sea demasiado costoso revelar su contenido. El mismo se basa en algoritmos de cifrado de clave simétrica. Uno de los principales inconvenientes que presenta este enfoque es que debe existir una vía para que ambas partes, cliente y servidor puedan conocer la llave, corriendo el riesgo de que si es enviada por la red esta pueda ser interceptada con las complicaciones que esto conlleva. En aras de mitigar este riesgo dicho mecanismo cuenta con una vía para la generación de dicha llave de forma local, tanto en el cliente como en el servidor.

Generación de la clave secreta

Para evitar el envío de la llave de cifrado a través de la red se ha diseñado un mecanismo para la generación de esta de manera local, en el cliente y en el servidor. Esta generación se basa principalmente en el hecho de que tanto cliente como servidor poseen una porción de la información de generación (semilla de generación) y que es generada aleatoriamente, en la definición de una estrategia de generación flexible que permite ser modificada en tiempo de ejecución y en la utilización de un parámetro aleatorio (número entero grande) que influya en el comportamiento de la estrategia de generación.

El primer paso es el intercambio por ambas partes de la información necesaria para generar la llave lo cual se realiza mediante un intercambio de mensajes donde el cliente es quién comienza la conversación enviando su parte de la semilla, en respuesta el servidor envía la otra parte, luego se envía hacia el cliente una cadena de texto especificando la estrategia de generación que ambos utilizarán y el parámetro aleatorio utilizado por la estrategia. Finalmente se procede a la generación de la llave secreta por parte de ambos. Las llaves generadas para cada cliente se almacenan en el servidor utilizando un identificador proporcionado por el cliente de manera que se pueda obtener la llave correcta para cada uno de ellos.

La estrategia de generación se especifica por medio de una cadena compuesta por los identificadores de los algoritmos que la conforman separados por un caracter “/”, por ejemplo “and/xor/rotBi/rotBy”. Cada uno de estos cuatro identificadores representa un algoritmo que se le aplicará a la semilla de generación anteriormente mencionada en el mismo orden en que aparecen en la cadena. La ventaja de este mecanismo es que modificando el orden de los algoritmos cambia completamente la salida generada, además se puede repetir arbitrariamente la aparición de cada algoritmo agregando complejidad al proceso de generación.

EL algoritmo basado en la operación and recorre los bytes que conforman la semilla de generación desde el principio, seleccionando dos bytes consecutivos (bi, bi+1) en cada pasada. El primero de estos es la base para la operación proporcionando sus 8 bits, el segundo solo proporciona un bit, el cual es elegido de acuerdo a un valor que se extrae del parámetro aleatorio y que indica la posición del bit que se debe seleccionar. En caso de que el número (n) que indica el bit que se debe seleccionar sea mayor que 6 (el séptimo bit no se utiliza ya que está reservado para el signo) entonces n = 9 – n. El resultado de esta operación reemplaza al primer byte en la semilla de generación.

EL algoritmo basado en la operación xor funciona de forma similar al algoritmo anterior en cuanto a la manera de recorrer la semilla de generación, la selección de los bytes procesados en cada pasada y la utilización de un valor aleatorio en el proceso. La diferencia radica en que la operación realizada es como indica su nombre un xor, donde esta vez si se toman todos los bits de cada uno de los bytes seleccionados. Al resultado de la operación entre los dos bytes se le realiza un xor nuevamente con el valor aleatorio y el resultado reemplaza al primer byte en la semilla de generación.

El algoritmo basado en una operación de rotación de bits (rotBi) recorre la semilla y va seleccionando los bytes uno a uno, así como un valor aleatorio. Luego al byte seleccionado se le realiza una rotación de sus

bits n posiciones a la derecha dependiendo del valor aleatorio seleccionado. El resultado de esta operación reemplaza al byte seleccionado en la semilla de generación.

El algoritmo basado en una operación de rotación de bytes (rotBy) actúa sobre toda la semilla de generación rotando todos los bytes presentes en la misma en cada pasada, basándose en dos valores aleatorios donde uno indica la cantidad de posiciones a desplazar en la rotación y el otro el sentido de la misma. Si el segundo valor aleatorio es par entonces la rotación será hacia la derecha, en caso contrario será a la izquierda.

2.4.5.2 Mecanismo de integridad

Dentro de esta propuesta de solución fue necesario definir e implementar un mecanismo que permitiera garantizar la integridad de la información que se transmite entre cliente y servidor. El mismo es responsable de comprobar que la información que llega al receptor es la misma que ha sido enviada, o sea, que no ha sufrido cambios en el trayecto. Con este objetivo, esta propuesta utiliza los algoritmos de resumen md5 y sha-1, los que permiten obtener una representación compacta de la información que se envía, de manera que aplicando el mismo algoritmo una vez que esta llegue a su destino ambos resúmenes puedan ser comparados verificando así la integridad de los datos recibidos como se puede apreciar en la *Fig.10.*

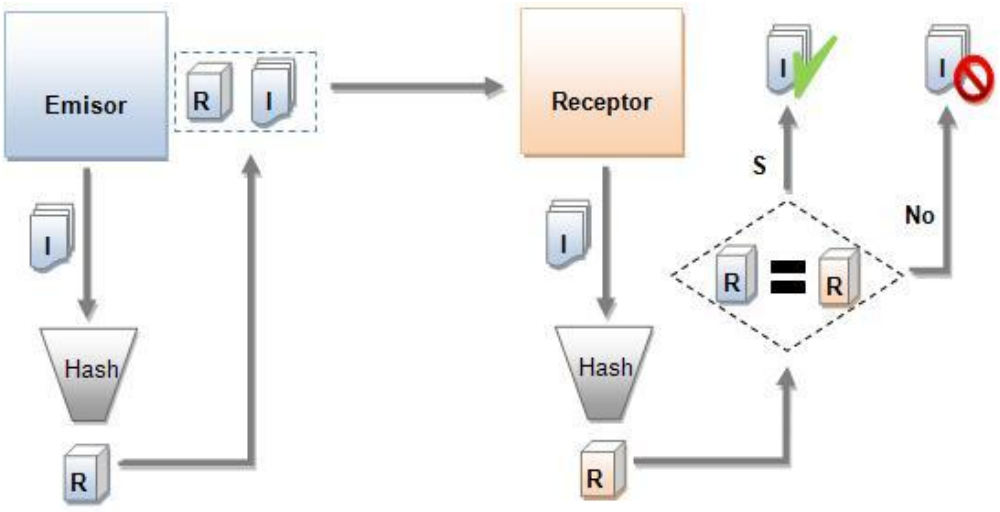


Fig. 10 Mecanismo para comprobar la integridad de la información utilizando funciones de resumen (Hash).

2.4.5.3 Abstracción

Uno de los principales retos que presenta la solución de software de esta investigación es el de lograr que el proceso de cifrado y descifrado así como el proceso de verificación de la integridad de la información sean transparentes para los clientes, permitiéndole abstraerse de las complejidades implícitas en los mismos. Para lograr esto es necesario proporcionar una vía para adicionar metadatos al código de las aplicaciones clientes, lo cual se logra mediante la utilización de las anotaciones @Confidencial, @Integral y @UbicacionRemota.

La anotación @Confidencial puede ser utilizada a nivel de clase o de método y su función es indicar si se requiere el cifrado de los datos que se envían hacia y desde el servidor. Esta anotación acepta un atributo que especifica cuándo se debe cifrar la información: al enviarla hacia el servidor, cuando el servidor responde, en ambos casos o nunca. *Ver Fig.11*.

```
@Confidencial(tipo = TipoAccion.RESPUESTA)
int calcularSuma(int n1, int n2);
```

Fig. 11 Utilización de la anotación @Confidencial

La anotación @Integral es similar a la anterior solo que esta indica si se requiere la verificación de integridad. Además acepta otro atributo que indica el algoritmo utilizado para calcular el resumen de los datos que se envían, por defecto es md5 y acepta también sha-1. *Ver Fig.12*.

```
@Integral(tipo = TipoAccion.ENVIO, resumen = "md5")
String obtenerSaludo(String nombreBean);
```

Fig. 12 Utilización de la anotación @Integral

La anotación @UbicacionRemota es utilizada a nivel de clase y acepta como atributo una URL mediante la cual se puede obtener una instancia de un objeto remoto utilizando el JNDI. Esta cadena debe tener el formato *nombre-modulo-ejb/nombre-bean!NC*. El elemento *nombre-modulo-ejb* se refiere al nombre del módulo EJB donde se encuentra el bean remoto, el elemento *nombre-bean* se refiere al nombre del bean remoto del que se requiere una instancia por último el elemento *NCI* es opcional y representa el nombre completo de la interfaz del bean que se desea obtener. *Ver Fig. 13*.

```
@UbicacionRemota(valor = "prueba-ejb/PruebaEjb")
@Confidencial(tipo = TipoAccion.SIEMPRE)
public interface PruebaEjbRemote {
 ...
}
```

Fig. 13 Utilización de la anotación @UbicacionRemota

Para obtener una instancia de un objeto remoto este mecanismo proporciona la clase *GestorObjetoRemoto* que cuenta con el método *T getObjetoRemoto (Class<T> interf)* el cual recibe como parámetro un objeto de tipo Class que representa la interfaz del objeto remoto del cual se solicita una instancia, y devuelve una instancia del objeto remoto solicitado.

Este método analiza el parámetro *interf* en busca de la anotación @Confidencial o @Integral en alguno de sus métodos o a nivel de la declaración de clase. Si no se encuentra quiere decir que no se requiere de una comunicación cifrada o de la verificación de integridad por lo que el objeto remoto es obtenido de manera convencional utilizando el JNDI y devuelto al cliente. En caso de encontrarse alguna de estas anotaciones entonces se genera en tiempo de ejecución un objeto proxy para la interfaz representada por *interf* que es devuelto al cliente. Este objeto proxy permite que cualquier método invocado en la interfaz remota sea interceptado (25) permitiendo de esta forma que los parámetros puedan ser cifrados, empaquetados y luego enviados al servidor. Una vez que los datos se encuentran en el servidor se realiza el proceso inverso, y se invoca el método requerido en el objeto remoto objetivo.

2.4.5.4 Encapsulamiento

Para lograr que el mecanismo de confidencialidad e integridad funcione adecuadamente, fue necesario definir un protocolo de paquetes que permita encapsular la información para lograr una correcta comunicación entre el cliente y el servidor.

Cada paquete está compuesto por una cabecera y los datos propiamente. La cabecera contiene información extra necesitada por el receptor para realizar distintas operaciones, como pueden ser el descifrado de los datos y la invocación de un método en un objeto remoto determinado.

El protocolo de paquetes define que en el nivel más bajo se encuentra el *Paquete de Invocación*, luego le sigue el *Paquete de Cifrado*, el de *Integridad* y por último el de *Paquete de Operaciones* como se muestra en la *Fig.14.*

Fig. 14 Jerarquía de paquetes.

El paquete más significativo es el de invocación, el cual contiene en su cabecera la URL del objeto remoto al que va dirigida la información, el nombre del método que se debe ejecutar en el objeto remoto y los tipos de datos de los argumentos que recibe dicho método. Los datos de este paquete son los argumentos que recibe el método remoto que debe invocarse y que constituye la información asegurada.

2.4.6 Activación de estaciones de trabajo

Con vista a lograr un mayor control sobre las estaciones de trabajo desde donde se ejecutan procedimientos remotos en el servidor de aplicaciones, fue necesario definir un mecanismo que permitiese a las aplicaciones que utilizan el marco de trabajo Kairos determinar si una estación cliente se encuentra o no autorizada para interactuar con la lógica de negocio publicada en el servidor.

Este mecanismo debe realizar el proceso de verificación basándose en algún elemento distintivo de una y sólo una estación cliente y que tenga un carácter invariable, de forma que no sea posible para alguna persona suplantar este elemento desde otra estación. Por lo que se propone que dicho elemento sea la dirección de Control de Acceso al Medio (MAC por sus siglas en inglés). Además podrían incluirse otros elementos representativos de la estación de trabajo que se desea activar como pueden ser la dirección IP

y el nombre de la estación. Estos elementos no deben ser tomados como identificadores ya que pueden ser modificados fácilmente pero se pueden utilizar con carácter informacional.

El proceso de activar una estación de trabajo debe ser el primer paso de una aplicación cliente para poder acceder a cualquiera de las funcionalidades que se encuentran desplegadas en el servidor. Además se recomienda no permitir a los usuarios comunes realizar este proceso si no que se definan usuarios con privilegios diferenciados para ello. Se debe contar también con una fuente de almacenamiento de información donde se encuentren registrados los datos de las estaciones de trabajo que pueden activarse así como un mecanismo para la gestión de estos datos, de manera que sea sencillo administrar las estaciones activadas y con permiso para hacerlo.

Debe tenerse en cuenta que obtener información como la MAC de la estación de trabajo influye en la portabilidad del marco de trabajo ya que esta operación es dependiente de la plataforma donde se encuentre ejecutándose el sistema, por lo que se recomienda implementar soluciones que permitan obtener esta información sobre las plataformas más utilizadas en el centro CEGEL.

2.5 Conclusiones

En este capítulo se realizó la preparación para la implementación de la solución de software de esta investigación por medio de las tarjetas CRC, lo que posibilita minimizar la cantidad de errores durante el proceso de codificación, además de proporcionar una guía para el mismo. Se presentaron los diagramas de clases de las diferentes librerías que componen esta solución, así como la descripción de la arquitectura de la solución, además de los patrones de diseño empleados en el desarrollo de la misma. Finalmente se definieron los mecanismos necesitados por el marco de trabajo Kairos para garantizar la seguridad de la información gestionada por las aplicaciones que lo utilicen.

Capítulo 3: Implementación y pruebas

3.1 Introducción

En el presente capítulo se describe el Modelo de Implementación y se muestran los estándares de codificación utilizados en el desarrollo de la solución de software de esta investigación, además se evalúa el grado de calidad y fiabilidad de los resultados obtenidos en la implementación mediante la realización de pruebas de unidad al código.

3.2 Modelo de Implementación

El Modelo de Implementación está formado por los Diagramas de Componentes y de Despliegue, el mismo describe como se implementan los elementos del Modelo de Diseño en términos de componentes y señala la dependencia que existe entre los mismos.

3.2.1 Diagrama de Componentes

Los Diagramas de Componentes son usados para estructurar el Modelo de Implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación. (23) Un simple componente puede ser manifestado por múltiples artefactos, los cuales pueden residir en uno o diferentes entornos de ejecución, entonces un componente puede ser indirectamente implementado en múltiples entornos.

Después de haber analizado el concepto de componente se muestra en la *Fig.15* el Diagrama de Componentes perteneciente al mecanismo de seguridad desarrollado para el marco de trabajo Kairos en la presente investigación así como la descripción de cada uno de ellos.

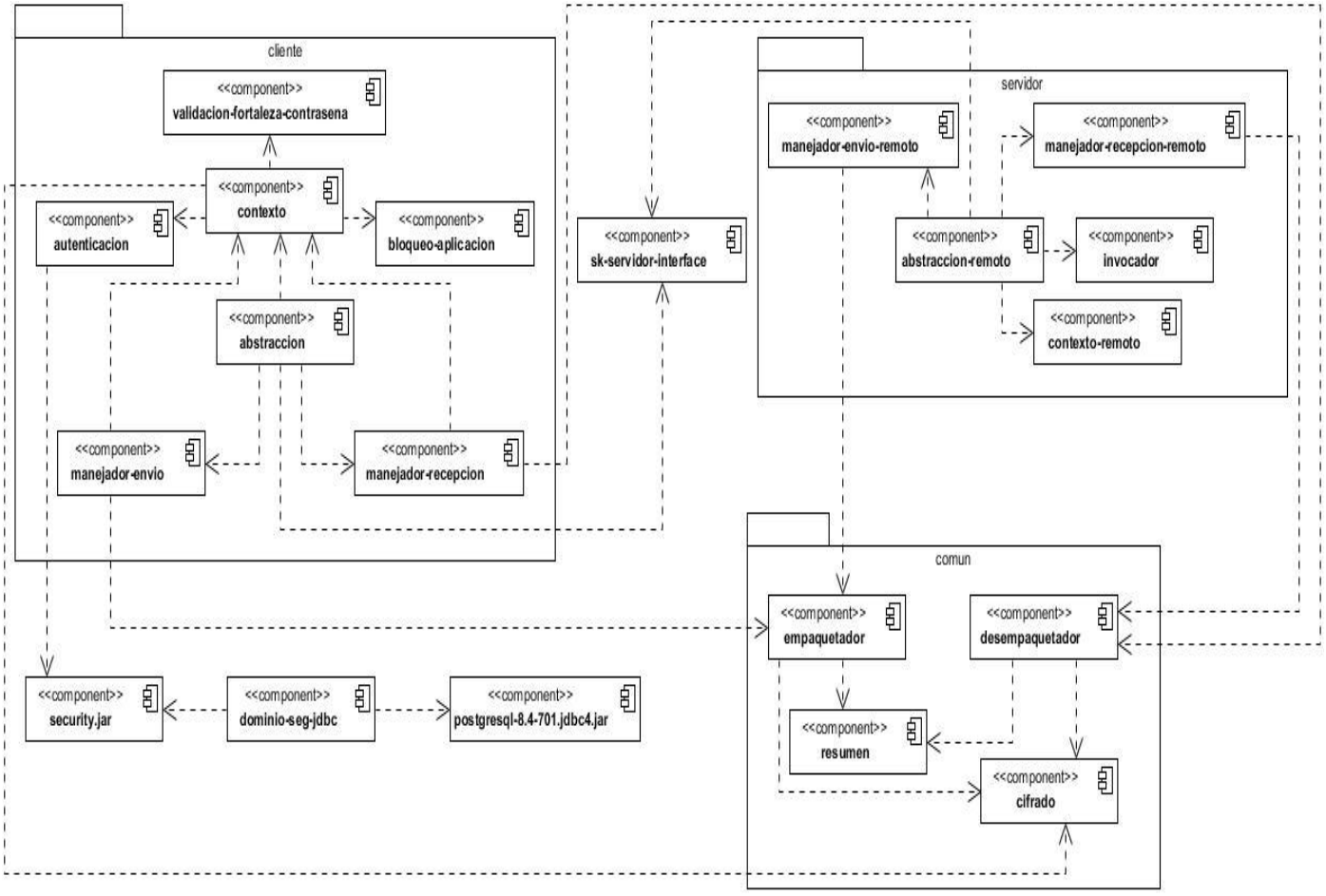


Fig. 15 Diagrama de Componentes.

Tabla 2: Componente *contexto*.

contexto	
Descripción	Componente encargado de inicializar el mecanismo de seguridad de acuerdo a la configuración recibida de parte del cliente.
Clases	ContextoSkCliente, ConstructorInitialContext, NegociadorLlaveSecreta

Tabla 3: Componente *autenticación*.

autenticacion	
Descripción	Componente encargado de realizar el proceso de autenticación contra el servidor de aplicaciones.
Interfaces	Autenticador
Clases	AutenticadorProgramatico

Tabla 4: Componente *validacion-fortaleza-contrasena*.

validacion-fortaleza-contrasena	
Descripción	Componente encargado de verificar si una contraseña posee un nivel de fortaleza especificado.
Interfaces	ExpresionFortaleza, ValidadorFortaleza
Clases	ExpresionFortalezaImpl, ValidadorFortalezaImpl

Tabla 5: Componente *bloqueo-aplicación*.

bloqueo-aplicación	
Descripción	Componente encargado de ejecutar una determina operación luego de un tiempo especificado en minutos sin que se detecte ningún evento de teclado ni ratón. Además de permitir detener o reiniciar el conteo del tiempo. Esta operación ejecutada puede ser el bloqueo de la aplicación cliente aunque podría ser cualquier otra.
Clases	Bloqueador

Tabla 6: Componente *abstracción*.

abstracción	
Descripción	Proporciona la abstracción necesaria para que los clientes de los mecanismos de integridad y confidencialidad no tengan que lidiar con las complejidades de los procesos ligados a los mismos, como el cifrado y la verificación de la integridad de la información que se intercambia entre cliente y servidor además del empaquetado y desempaquetado de esta para permitir una correcta comunicación. Este componente permite a los clientes resolver instancias de objetos remotos de una forma sencilla.
Anotaciones	Confidencial, Integral, UbicacionRemota
Interfaces	GestorSolicitud
Clases	InterceptorInvocacion, InterceptorInvocacionImpl, InstanciadorRemoto, InstanciadorProxy, InstanciadorObjetoRemoto, GestorSolicitudImpl

Tabla 7: Componente *manejador-envio*.

manejador-envio	
Descripción	Se encarga de procesar la información que se envía al servidor. Esto trae consigo el análisis del tipo de proceso requerido (confidencialidad y/o integridad), además se encarga de encapsular la información en paquetes para garantizar una correcta comunicación.
Interfaces	ManejadorEnvio
Clases	ManejadorEnvioConfidencialidad, ManejadorEnvioIntegridad

Tabla 8: Componente *manejador-recepcion*.

manejador-recepcion	
Descripción	Se encarga de procesar la información que se recibe del servidor. Para esto desencapsula los paquetes recibidos y luego analiza que tipo de proceso se requiere (confidencialidad y/o integridad).
Interfaces	ManejadorRecepcion
Clases	ManejadorRecepcionConfidencialidad, ManejadorRecepcionIntegridad

Tabla 9: Componente *contexto-remoto*.

contexto-remoto	
Descripción	Contiene la configuración del mecanismo en el servidor. Además se encarga de almacenar las llaves de cada cliente en cache, de manera que cuando se requiera se puedan obtener fácilmente teniendo el identificador del cliente.
Interfaces	ContextoSkServidorLocal
Clases	ContextoSkServidor, LlaveCache, CargadorRecursoConfig, NegociadorLlaveSecreta

Tabla 10: Componente *abstraccion-remoto*.

abstraccion-remoto	
Descripción	Se encarga de la comunicación con la parte cliente y de manejar los procesos de integridad y/o confidencialidad para lograr una comunicación segura.

Interfaces	GestorSolicitudLocal
Clases	GestorSolicitud, ManejadorComunicacionSeg

Tabla 11: Componente *manejador-envio-remoto*.

manejador-envio-remoto	
Descripción	Similar al manejador de envío presente en el cliente.
Interfaces	ManejadorEnvio
Clases	ManejadorEnvioIntegridad, ManejadorEnvioConfidencialidad

Tabla 12: Componente *manejador-recepcion-remoto*.

manejador-recepcion-remoto	
Descripción	Similar al manejador de recepción presente en el cliente.
Interfaces	ManejadorRecepcion
Clases	ManejadorRecepcionIntegridad, ManejadorRecepcionConfidencialidad, ManejadorInvocacion

Tabla 13: Componente *invocador*.

invocador	
Descripción	Se encarga de la invocación de métodos remotos utilizando metadatos como el nombre del objeto remoto que posee el método, el nombre del

	método y los tipos de los argumentos que recibe.
Interfaces	GestorInstanciaJndiLocal, Invocacion
Clases	GestorInstanciaJndi, InvocacionReflect

Tabla 14: Componente *empaquetador*.

empaquetador	
Descripción	Se encarga de encapsular la información en la estructura de paquetes determinada.
Interfaces	Empaquetador
Clases	EmpaquetadorConfidencialidad, EmpaquetadorIntegridad, EmpaquetadorInvocacion

Tabla 15: Componente *desempaquetador*.

desempaquetador	
Descripción	Se encarga de desencapsular la información de la estructura de paquetes determinada.
Interfaces	Empaquetador
Clases	EmpaquetadorConfidencialidad, EmpaquetadorIntegridad, EmpaquetadorInvocacion

Tabla 16: Componente *resumen*.

resumen	
Descripción	Contiene las clases relacionadas con el cálculo de resumen utilizando algoritmos hash.
Interfaces	Resumen
Clases	ResumenDecorador, ResumenMd5, ResumenSha1, FactoriaResumen

Tabla 17: Componente *cifrado*.

cifrado	
Descripción	Contiene las clases relacionadas con el cifrado de la información. Dentro de estas se incluyen las clases relacionadas con la creación de la llave de cifrado.
Interfaces	Cifrador, Estrategia
Clases	Cifrador3DES, EstrategiaBase, EstrategiaNula, EstrategiaAnd, EstrategiaXor, EstrategiaRotaBits, EstrategiaRotaBytes, FactoriaEstrategia

Tabla 18: Componente *sk-servidor-interface*.

sk-servidor-interface	
Descripción	Contiene las interfaces remotas que implementan los objetos remotos con los que se comunica el cliente.
Interfaces	ManejadorComunicacionSegRemote, NegociadorLlaveSecretaRemote

Tabla 19: Componente *dominio-seg-jdbc*.

dominio-seg-jdbc	
Descripción	Es el dominio de seguridad que se instala al servidor GlassFish Open Source Edition 3.0.1 para realizar la autenticación contra una Base de Datos.
Interfaces	DominioSeguridadBD, ModuloInicioSesionJaas, AdminMensaje

Tabla 20: Componente *security.jar*.

security.jar	
Descripción	Librería que provee el servidor de aplicaciones GlassFish Open Source Edition 3.0.1 donde se encuentran las clases bases para la implementación de un dominio de seguridad, además contiene la clase ProgrammaticLogin que permite realizar la autenticación desde el cliente.

Tabla 21: Componente *postgresql-8.4-701.jdbc4.jar*.

postgresql-8.4-701.jdbc4.jar	
Descripción	Driver de postgres para la conexión con la Base de Datos.

3.2.2 Diagrama de Despliegue

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. Un nodo a su vez, es un recurso de ejecución tal como un computador, un dispositivo o memoria. A continuación se muestra en la *Fig.16* el Diagrama de Despliegue para el mecanismo de seguridad desarrollado en la presente investigación para el marco de trabajo Kairos.

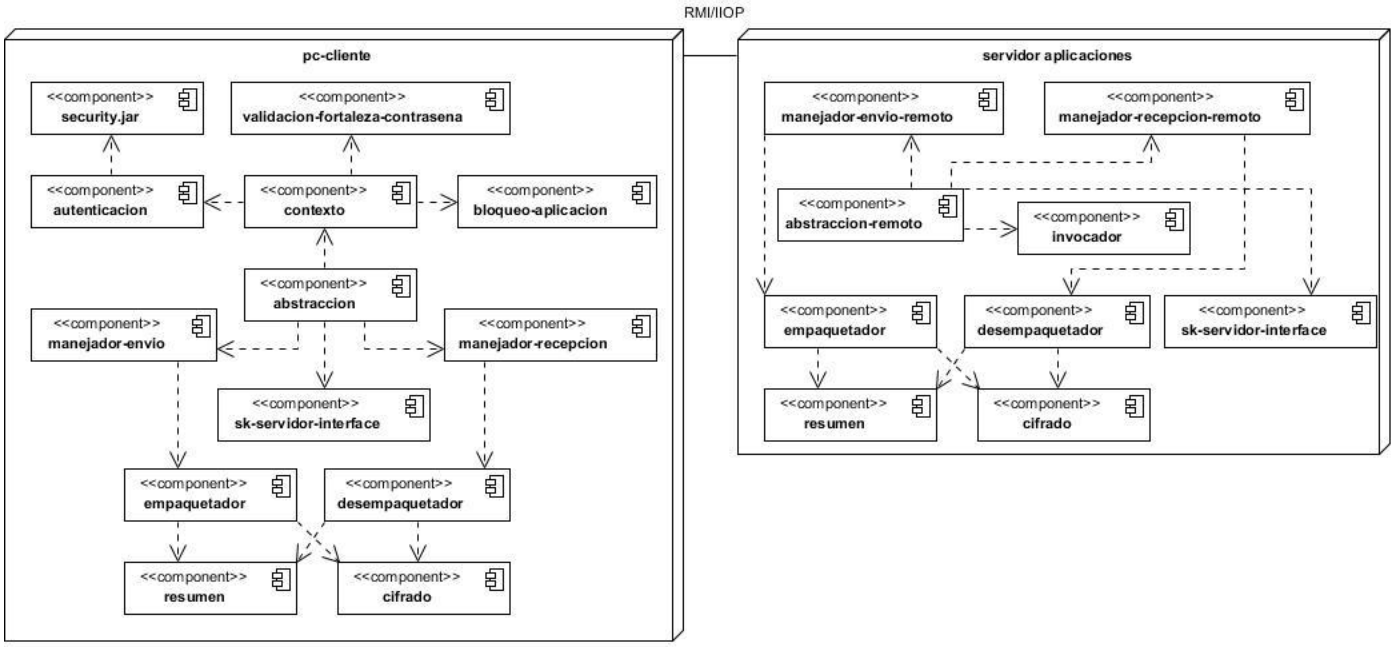


Fig. 16 Diagrama de Despliegue.

Como se puede apreciar el mecanismo de seguridad de la presente investigación se encontrará desplegado en dos nodos físicos, el cliente y el servidor, los que se comunican por medio del protocolo de invocación de método remoto de Java RMI/IIOP. En la parte cliente se encuentran los componentes relacionados con la autenticación, la validación de la fortaleza de las contraseñas, el bloqueo por inactividad así como la integridad y confidencialidad de la información, el componente de abstracción encargado de la interacción con los objetos clientes, y los manejadores de invocación.

En el servidor se encuentran el componente de abstracción que se encarga de la comunicación con la parte cliente, un componente encargado de invocar los procedimientos remotos requeridos así como los manejadores de invocación. Además en ambos nodos se encuentran componentes comunes como los encargados del empaquetamiento, desempaquetamiento, cifrado y resumen de la información así como las interfaces de los objetos remotos.

3.3 Estándares de codificación

Los estándares de codificación son un conjunto de reglas que se debe seguir a la hora de implementar las clases e interfaces del sistema. Además permiten un mayor entendimiento en el código de la aplicación. A continuación se muestran los estándares de codificación utilizados para el desarrollo de la solución de software de la presente investigación.

Declaraciones de clases e interfaces

La siguiente lista describe las partes de la declaración de una clase o interfaz, en el orden en que deberían aparecer.

- Comentario de documentación de la clase o interfaz (/**...*/).
- Sentencia class o interface.
- Comentario de implementación de la clase o interfaz si fuera necesario (/*...*/). Este comentario debe contener cualquier información aplicable a toda la clase o interfaz que no era apropiada para estar en los comentarios de documentación de la clase o interfaz.
- Variables de clase (static): Primero las variables de clase public, luego las protected, a continuación las de nivel de paquete (sin modificador de acceso), y por último las private.
- Variables de instancia: Primero las public, luego las protected, a continuación las de nivel de paquete (sin modificador de acceso), y por último las private.
- Constructores.
- Métodos: Los métodos se deben agrupar por funcionalidad más que por visión o accesibilidad. Por ejemplo, un método de clase privado puede estar entre dos métodos públicos de instancia. El objetivo es hacer el código más legible y comprensible.

Convenciones de nombres

Las convenciones de nombres son las que hacen los programas más entendibles haciéndolos más fácil de leer. También pueden dar información sobre la función de un identificador, por ejemplo, cuando es un paquete, o una clase, que puede ser útil para entender el código.

- ✓ **Paquetes:** El nombre de un paquete se escribe siempre con letras ASCII en minúsculas.

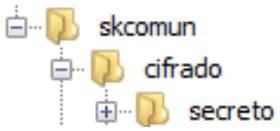


Fig. 17 Ejemplo de nombre de paquete.

- ✓ **Clases**: Los nombres de las clases deben ser sustantivos y cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Deben ser simples y descriptivos y estar compuestos por palabras completas, evitando acrónimos y abreviaturas.

```
/**...*/
public class ManejadorEnvioIntegridad
```

Fig. 18 Ejemplo de nombre de clase.

- ✓ **Interfaces:** Los nombres de las interfaces siguen la misma regla que las clases.

```
/**...*/
public interface GestorSolicitud
```

Fig. 19 Ejemplo de nombre de interfaz.

- ✓ **Métodos*:*** Los métodos deben ser verbos y siempre deben empezar con minúscula. Además cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

```
/**...*/
public void adicionarEspecificacion(String proc, TipoAccion ta)
```

Fig. 20 Ejemplo de nombre de método.

- ✓ **Variables:** Los nombres de variables e instancias deben empezar con minúscula, excepto las constantes. En caso de ser compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula. Los nombres de variables no deben empezar con los caracteres subguión "_" o signo del dólar "$", aunque ambos están permitidos por el lenguaje, los mismos deben ser cortos pero con significado. La elección del nombre de una

variable debe ser un mnemónico, designado para indicar a un observador casual su función. Los nombres de variables de un sólo carácter se deben evitar, excepto para variables índices temporales. Los nombres comunes para variables temporales son i, j, k, m, y n para enteros; c, d, y e para caracteres.

```
private long idCliente;
private String algResumen;
```

Fig. 21 Ejemplo de nombre de variables.

- ✓ **Constantes:** Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión ("_").

```
/**...*/
public static String URL_OBJETO_REMOTO = "objeto_receptor";
```

Fig. 22 Ejemplo de nombre de constantes.

3.4 Modelo de pruebas

Las pruebas de software se realizan con el objetivo de verificar que las funcionalidades del sistema se han implementado correctamente y obtener de esta forma un producto de gran calidad. Para esto se tratan de diseñar pruebas que tengan la mayor probabilidad de encontrar el mayor número de errores con la mínima cantidad de tiempo y de esfuerzo.

3.4.1 Pruebas de unidad

Uno de los pilares de la metodología XP es el uso de las pruebas para comprobar el funcionamiento de los códigos que se vayan implementando. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección.

Es por esta razón que durante el desarrollo de esta solución se realizaron constantemente pruebas de unidad a las funcionalidades implementadas utilizando el marco de trabajo JUnit en su versión 4.5, permitiendo realizar un examen minucioso de los detalles procedimentales, comprobando los caminos lógicos del programa, comprobando los bucles y condiciones, y examinado el estado del programa en

varios puntos. A continuación se muestran ejemplos del código de prueba y los resultados de las mismas realizadas a las funcionalidades relacionadas con la generación de la llave de cifrado ya que es uno de los procesos críticos de esta solución, aunque se le realizaron pruebas a todos los métodos presentes en la solución. Los resultados de las mismas se pueden consultar en el Anexo4.

Finalmente se muestran los resultados de las tres iteraciones de las pruebas de unidad realizadas a las funcionalidades implementadas. Tabla 23, 24 y 25.

EstrategiaAnd

El método **and** realiza esta operación lógica entre su primer argumento y un bit seleccionado del segundo argumento. El tercer argumento especifica cuál es el bit que se debe seleccionar.

```
@Test
public void testAnd() {

    byte b1 = 127;//0111 1111
    byte b2 = 0;  //0000 0000
    byte masc = 6; //se selecciona el bit 6 de b2
    EstrategiaAnd instance = new EstrategiaAnd();
    byte expResult = 63;//0011 1111
    byte result = instance.and(b1, b2, masc);
    assertEquals(expResult, result);
}

/**
 * Test of procesar method, of class EstrategiaAnd.
 */
@Test
public void testProcesar() {

    byte[] dato = "1234567890".getBytes();
    Object[] ctx = new Object[]{new byte[]{1,8,0,6,3,1,4,7,9,6,5}};
    EstrategiaAnd instance = new EstrategiaAnd();
    instance.sgte = new EstrategiaNula();
    byte[] result = instance.procesar(dato, ctx);
    assertNotNull(result);
}
```

Fig. 23 Código de prueba de los métodos de la clase EstrategiaAnd.

Durante la primera iteración de pruebas a esta clase se detectó un fallo en el algoritmo, lo cual traía consigo que el resultado del método **and** no fuese el que se esperaba como se puede apreciar en el reporte generado para esta prueba que se muestra en la *Fig. 24.*

Class skcomun.cifrado.secreto.EstrategiaAndTest

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
EstrategiaAndTest	2	0	1	0.072	2012-05-15T14:11:25	dss-PC

Tests

Name	Status	Type	Time(s)
testAnd	Failure	expected:<63> but was:<31> junit.framework.AssertionFailedError: expected:<63> but was:<31> at skcomun.cifrado.secreto.EstrategiaAndTest.testAnd(EstrategiaAndTest.java:47)	0.007
testProcesar	Success		0.001

Fig. 24 Reporte generado para la prueba realizada a la clase EstrategiaAnd.

Luego de detectar la falla se corrigió y se ejecutó nuevamente la prueba para los mismos valores arrojando un resultado satisfactorio como se puede apreciar en el nuevo reporte generado.

Class skcomun.cifrado.secreto.EstrategiaAndTest

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
EstrategiaAndTest	2	0	0	0.061	2012-05-15T14:53:56	dss-PC

Tests

Name	Status	Type	Time(s)
testAnd	Success		0.002
testProcesar	Success		0.000

Fig. 25 Nuevo reporte generado para la prueba realizada a la clase EstrategiaAnd.

EstrategiaXor

El método **xor** realiza la operación lógica indicada por su nombre entre sus dos primeros argumentos, luego realiza nuevamente esta operación entre el resultado de la primera y un byte cuyo único bit en 1 es el que se encuentra en la posición especificada por el tercer argumento.

```
/**...*/
@Test
public void testXor() {

    byte b1 = 15; //0000 1111
    byte b2 = 112;//0111 0000
    byte masc = 1;
    EstrategiaXor instance = new EstrategiaXor();
    byte expResult = 125;//0111 1101
    byte result = instance.xor(b1, b2, masc);
    assertEquals(expResult, result);
}

/**...*/
@Test
public void testProcesar() {

    byte[] dato = "1234567890".getBytes();
    Object[] ctx = new Object[]{new byte[]{1, 8, 0, 6, 3, 1, 4, 7, 9, 6, 5}};
    EstrategiaXor instance = new EstrategiaXor();
    instance.setSgte(new EstrategiaNula());
    byte[] result = instance.procesar(dato, ctx);
    assertNotNull(result);
}
```

Fig. 26 Código de prueba de los métodos de la clase EstrategiaXor.

Las pruebas realizadas a las funcionalidades de esta clase luego de varias iteraciones arrojaron resultados satisfactorios como muestra el reporte generado que se muestra a continuación.

Class skcomun.cifrado.secreto.EstrategiaXorTest

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
EstrategiaXorTest	2	0	0	0.062	2012-05-15T20:30:16	dss-PC

Tests

Name	Status	Type	Time(s)
testXor	Success		0.003
testProcesar	Success		0.001

Fig. 27 Reporte generado para la prueba realizada a la clase EstrategiaXor.

EstrategiaRotaBits

El método **rotar** realiza una rotación a la izquierda de los bits de su primer argumento la cantidad de posiciones indicadas por el segundo argumento.

```
/**...*/
@Test
public void testRotar() {

    byte b = 15;//0000 1111
    byte rot = 3;//rota 3 bits a la derecha
    EstrategiaRotaBits instance = new EstrategiaRotaBits();
    byte expResult = 113;//0111 0001
    byte result = instance.rotar(b, rot);
    assertEquals(expResult, result);
}

/**...*/
@Test
public void testProcesar() {

    byte[] dato = "1234567890".getBytes();
    Object[] ctx = new Object[]{new byte[]{1, 8, 0, 6, 3, 1, 4, 7, 9, 6, 5}};
    EstrategiaRotaBits instance = new EstrategiaRotaBits();
    instance.setSgte(new EstrategiaNula());
    byte[] result = instance.procesar(dato, ctx);
    assertNotNull(result);
}
```

Fig. 28 Código de prueba de los métodos de la clase EstrategiaRotaBits.

Las pruebas realizadas a las funcionalidades de esta clase luego de varias iteraciones arrojaron resultados satisfactorios como muestra el reporte generado que se muestra a continuación.

Class skcomun.cifrado.secreto.EstrategiaRotaBitsTest

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
EstrategiaRotaBitsTest	2	0	0	0.073	2012-05-15T20:30:15	dss-PC

Tests

Name	Status	Type	Time(s)
testRotar	Success		0.003
testProcesar	Success		0.001

Fig. 29 Reporte generado para la prueba realizada a la clase EstrategiaRotaBits.

EstrategiaRotaBytes

El método procesar realiza una rotación de los bytes presentes en el primer argumento de acuerdo a un conjunto de bytes que representan un patrón de rotación. Este patrón indica la cantidad de posiciones y la dirección de la rotación.

```
/**...*/
@Test
public void testProcesar() {

    byte[] dato = "1234567890".getBytes();
    Object[] patron = new Object[]{new byte[]{1, 8, 2, 6, 3, 1, 4, 7, 9, 6, 5}};
    EstrategiaRotaBytes instance = new EstrategiaRotaBytes();
    instance.setSgte(new EstrategiaNula());
    byte[]expected = "3456789012".getBytes();
    byte[] result = instance.procesar(dato, patron);
    assertArrayEquals(expected, result);
}
```

Fig. 30 Código de prueba de los métodos de la clase EstrategiaRotaBytes.

Luego de realizar varias pruebas a esta funcionalidad y corregir los errores detectados se obtuvo un resultado satisfactorio como lo muestra el reporte que se muestra a continuación.

Class skcomun.cifrado.secreto.EstrategiaRotaBytesTest

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
EstrategiaRotaBytesTest	1	0	0	0.067	2012-05-15T20:30:16	dss-PC

Tests

Name	Status	Type	Time(s)
testProcesar	Success		0.004

Fig. 31 Reporte generado para la prueba realizada a la clase EstrategiaRotaBytes.

GeneradorLlave

Por último se realiza una prueba de la generación de la llave secreta en general donde se verifica la correcta creación de la misma.

```
@Test
public void testGeneracionLlaveSecreta() {

    byte[]paramGen = Utilidad.getArrDigitos(new Date().getTime());
    byte[] semilla = new byte[24];
    genSemilla(semilla);
    SecretKey llave = null;
    try {
        Estrategia estrategia = FactoriaEstrategia.getEstrategia("xor/and/rotBy/rotBi");
        Assert.assertNotNull(estrategia);

        llave = GeneradorLlave.generarLlave(semilla, estrategia,
                new Object[]{paramGen}, ConstanteComun.TRIPLE_DES);
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
    Assert.assertNotNull(llave);
}

private void genSemilla(byte[] semilla) {
    SecureRandom sr = new SecureRandom();
    sr.nextBytes(semilla);
}
```

Fig. 32 Código de prueba de los métodos de la clase GeneradorLlave.

Primeramente se genera una estrategia por medio de la cadena que representa los algoritmos que la componen y se verifica que la estrategia se haya creado correctamente, luego se genera la llave utilizando una semilla de generación, la estrategia, un parámetro aleatorio y el algoritmo de cifrado, por último se verifica que la llave haya sido creada.

Luego de varias iteraciones de pruebas y de haber corregido las deficiencias detectadas se obtuvo un resultado satisfactorio al realizar esta prueba lo cual se puede apreciar en el reporte que se muestra a continuación.

Class test.GenLlaveTest

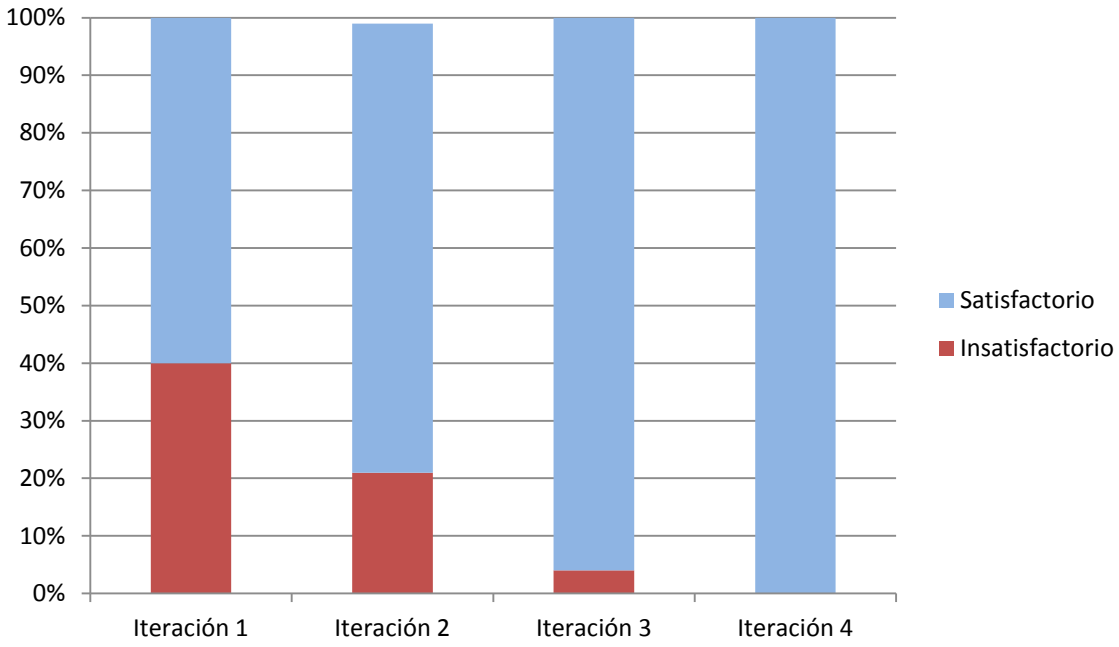
Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
GenLlaveTest	1	0	0	0.229	2012-05-15T12:32:23	dss-PC

Tests

Name	Status	Type	Time(s)
testGeneracionLlaveSecreta	Success		0.173

Fig. 33 Reporte generado para la prueba realizada a la clase GeneracionLlaveSecreta.

Los resultados de las diferentes iteraciones de pruebas se pueden apreciar en la gráfica que se muestra a continuación.



Gráfica 1. Resultados de las pruebas de unidad

3.4.2 Pruebas de aceptación

Antes de entregar un producto de software al cliente se deben realizar pruebas de aceptación o caja negra con el objetivo de probar que lo que se ha desarrollado realiza lo que espera el cliente. En la solución de software desarrollada como parte de la presente investigación estas pruebas fueron realizadas de manera conjunta con la aplicación SIGESAP donde se utilizó el marco de trabajo Kairos. Los resultados de estas pruebas constan en las actas de aceptación realizadas por la empresa Calisoft la cual cuenta con personal capacitado para realizar este tipo de pruebas.

3.5 Conclusiones

En este capítulo se realizaron los Diagramas de Componente y de Despliegue, y se especificaron los estándares de codificación definidos por el equipo de desarrollo para la implementación de la solución. Además se realizó la validación de la implementación por medio de las pruebas unitarias y las pruebas de

aceptación dando de esta forma por terminado el desarrollo de un mecanismo de seguridad para el marco de trabajo Kairos.

Conclusiones generales

El presente trabajo se enfocó en el estudio y desarrollo de un mecanismo de seguridad para el marco de trabajo Kairos que permitiera asegurar la información manejada por las aplicaciones que lo utilizan, realizando para esto un estudio del arte de un conjunto de herramientas, tecnologías y metodologías que permitió seleccionar las que se adecuaran mejor al objetivo de la investigación.

Además se realizó un estudio de algunos elementos de seguridad necesarios en muchas de las aplicaciones de software actuales lo cual permitió definir los mecanismos de seguridad necesarios por el marco de trabajo Kairos para posteriormente ser diseñados, obteniendo artefactos como el Diagrama de clases de cada una de las librerías que conforman la solución de software de esta investigación, el Diagrama de Componentes así como el Diagrama de Despliegue. Finalmente se validó la solución de software por medio de las pruebas de unidad y aceptación, comprobando de esta forma que se logró cumplir con el objetivo de la investigación con calidad y cumpliendo con los resultados esperados, aportando beneficios como:

- ✓ Ahorro en tiempo y recursos para el equipo de desarrollo del marco de trabajo Kairos.
- ✓ Reutilización de código y componentes en proyectos con características similares.
- ✓ Aseguramiento de la información manejada por las aplicaciones que utilizan el marco de trabajo Kairos.

Recomendaciones

Debido a los resultados de la investigación efectuada y de la experiencia adquirida durante la realización de este trabajo, y con el propósito de asegurar la posterior ampliación, modificación y mejora del mecanismo de seguridad propuesto, se exponen a continuación algunas recomendaciones:

- ✓ Continuar con el desarrollo de este mecanismo dotándolo de nuevas funcionalidades.
- ✓ Adicionar un mecanismo de cifrado basado en clave asimétrica que brinde mayor seguridad que el actualmente utilizado de clave simétrica.
- ✓ Adicionar un mecanismo para el control de excepciones.

Bibliografía

1. Segu-Info. [En línea] http://www.segu-info.com.ar/logica/seguridadlogica.htm.

2. Segu-Info. [En línea] http://www.segu-info.com.ar/logica/identificacion.htm.

3. **Goncalves, Antonio.** *Beginning Java EE 6 Platform With GlassFish 3.* s.l. : Apress, 2010.

4. **Rivest, R.** ietf. [Online] http://www.ietf.org/rfc/rfc1321.txt.

5. **Xuejia Lai, Kefei Chen.** *Finding SHA-1 Characteristics: General Results and Applications.* 2006.

6. **Coulouris, George.** *Sistemas Distribuidos. Tercera Edición.* Madrid : Addison Wesley, 2001.

7. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software.* Madrid : Pearson Addison-Wesley, 2000. 84-7829-036-2.

8. *Metodologías Tradicionales Vs. Metodologías Ágiles.* **Figueroa, Roberth G., Solís, Camilo J. y Cabrera, Armando A.** Loja : Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación, 2007.

9. **Beck, Kent.** *Extreme Programming Explained: Embrace Change.* 1999.

10. **Robert C. Martin, James W. Newkirk.** *Extreme Programming in Practice.* 2001.

11. **Wake, William C.** *Extreme Programming Explored.* s.l. : Addison-Wesley Professional, 2001.

12. **Collado Cabeza, Eduardo y Díaz Berenguer, Angi.** [En línea] Madritel, 2003. http://web.madritel.es/personales3/edcollado/index.html.

13. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Lenguaje Unificado de Modelado.* s.l : Pearson Addison-Wesley, 1999. 84-7829-028-1.

14. **Menéndez, Rosa.** Proyectos. [En línea] 2009. http://www.usmp.edu.pe/publicaciones/boletin/fia/info36/proyectos.html#a.

15. **Universidad de Oviedo.** Sitio Web de la E.U de Ingeniería Técnica Informática de Oviedo. *Sitio Web de la E.U de Ingeniería Técnica Informática de Oviedo.* [En línea] 2003. http://petra.euitio.uniovi.es/.

16. **Kalali, Masoud.** *GlassFish Security .* Birmingham : Packt Publishing Ltd, 2010 . p. 74.

17. **Corporation, Oracle.** *Oracle GlassFish Server 3.0.1 Development Guide.* 2010.

18. **Andrew Lee Rubinger, Bill Burke.** *Enterprise JavaBeans 3.1.* s.l. : Oreilly, 2010.

19. **E. A. Bertino, L. A. Martino.** *Sistemas de bases de datos orientados a objetos.* 1995.

20. **Casanova, Jaime y Herrera, Álvaro.** *MVCC.* [presentación] Ciudad de la Habana : s.n, 2009.

21. **Sanchez Medrano, Jesus Rafael.** *Introducción a Base de Datos con PostgreSQL.* 2009.

22. **Oracle.** Oracle. [En línea] http://www.oracle.com/us/products/database/overview/index.html.

23. **Pressman, Roger S.** *Ingeniería de software un enfoque práctico.* s.l : Mcgraw-hil, 2001.

24. **Garlan, David y Shaw, Mary.** *An introduction to software architecture.* s.l. : CMU, 1994.

25. **Ira Foreman, Nate Foreman.** *Java Reflection in Action.* s.l. : Manning, 2005.

26. **Colectivo de autores.** Sitio Web oficial Visual-Paradigm. [En línea] Visual Paradigm Organización, 2009. http://www.visual-paradigm.com.

27. *Java Reflection in ActionGreenwichManning.* 2005.

28. **Kuchana, Partha.** *Software Architecture Design Patterns in Java.* s.l. : Auerbach, 2004. p. 23.

29. **Grupo soluciones GSINova.** [En línea] http://www.rational.com.ar/herramientas/roseenterprise.html.

30. **Pressman, Roger S.** *Ingeniería de Software un enfoque práctico.* 1997. Quinta Edición.

31. **Pressman, Roger S.** *Ingeniería de Software un enfoque práctico.* 2005. Sexta Edición.

32. **Gómez, Omar.** Especificación e implementación de software basado en componentes con la tecnología EJB 3.0. [En línea] Enero 2008. http://osgg.net/omarsite/resources/papers/cbse_ejb.pdf. 22.

33. **Keith, Mike y Schincariol, Merrick.** *Pro JPA2 Mastering the Java Persistence API.* New York : Springer -Verlag, 2009. 978-1-4302-1956-9.

34. **Group, PostgSQL Global Development.** *PostgreSQL 8.4.1 Documentation.* California : s.n, 2009. 15.

35. **Oracle Corporation .** NetBeans IDE 6.9 Release Information. [En línea] 2010.

36. **Eclipse.** Inc. Eclipse Documentation - Current Release (Eclipse Galileo). Help Eclipse SDK. [En línea] http://help.eclipse.org/galileo/index.jsp .

37. **Oracle Corporation .** Oracle Sun Developer Network (SDN). GlassFish Community. [En línea] https://glassfish.dev.java.net/.

38. **González Cornejo, José Enrique.** Arquitectura en Capas. [En línea] http://www.docirs.cl/Arquitectura_Tres_Capas.htm.

39. **Valle, José.** Arquitectura Cliente-Servidor. [En línea] http://www.monografias.com/trabajos24/arquitectura-cliente-servidor/arquitectura-cliente-servidor.shtml.

40. **Booch, Grady, Rumbaugh, James y Jacobson, Ivar.** *El lenguaje unificado de modelado.* s.l. : Pearson Addison-Wesley, 2007. Segunda Edición.

Glosario de términos

Mnemónico: En informática, un mnemónico es una palabra que sustituye a un código de operación (lenguaje de máquina), con lo cual resulta más fácil la programación, es de aquí de donde se aplica el concepto de lenguaje ensamblador.

Un lenguaje en mnemónico o lista de instrucciones consiste en un conjunto de códigos simbólicos, cada uno de los cuales corresponde a una instrucción. Cada fabricante utiliza sus propios códigos, y una nomenclatura distinta para nombrar las variables del sistema. El lenguaje en mnemónico es similar al lenguaje ensamblador del micro.

Control de Acceso al Medio (MAC por sus siglas en inglés): Es un identificador de 48 bits que corresponde de forma única a una tarjeta o dispositivo de red.

Interfaz de Programación de Aplicaciones (API por sus siglas en inglés): Es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objeto (POO por sus siglas en inglés)) que ofrece cierta biblioteca para ser utilizado por otros software como una capa de abstracción. Son usadas también en las bibliotecas (también denominadas vulgarmente ¨librerías¨).

Enterprise Java Beans (EJB): Es una tecnología que forma parte de la especificación de la plataforma JEE 6 y proporciona una serie de servicios como inyección de instancias, manejo de transacciones y manejo de seguridad.

Bean: Un bean es un componente JEE6 que implementa la tecnología EJB y se ejecutan en un contenedor EJB, el cual es un entorno de ejecución dentro del servidor de aplicaciones.

Notación para el Modelado de Procesos de Negocio (BPMN por sus siglas en inglés): Es una notación gráfica que describe la lógica de los pasos de un proceso de Negocio. Esta notación ha sido especialmente diseñada para coordinar la secuencia de los procesos y los mensajes que fluyen entre los participantes de las diferentes actividades.

Interfaz de Nombrado y Directorio Java (JNDI por sus siglas en inglés): Es parte de la plataforma Java, ofreciendo a las aplicaciones basadas en tecnología Java una interfaz unificada para nombrado múltiple y servicios de directorio.