PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

PERFIL UML PARA DESARROLLO DE SISTEMAS SEGUROS

GUIDO ALEJANDRO ASIS ALBORNOZ

TESIS DE GRADO
MAGÍSTER EN INGENIERÍA INFORMÁTICA

(DICIEMBRE 2008)

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

PERFIL UML PARA DESARROLLO DE SISTEMAS SEGUROS

GUIDO ALEJANDRO ASIS ALBORNOZ

Profesor Guía: **Claudio Cubillos Figueroa**

Programa: **Magíster en Ingeniería Informática**

(Diciembre 2008)

RESUMEN

La seguridad es un aspecto de la ingeniería de software al cual no se le ha dado la importancia que tiene en el ciclo de desarrollo de software, ya que se le ha considerado como algo que se verifica al final del proceso. Con el tiempo esta forma de enfrentar la seguridad ha ido cambiando, comenzado a considerarla como una característica que se puede agregar al momento de la codificación, lo cual aún no es suficiente, ya que idealmente esta debería modelarse desde la etapa de diseño, teniendo que ser capaces de realizar las revisiones correspondientes en cada etapa. Considerando esto, se ha comenzado a integrar y a pensar en la seguridad en etapas cada vez más tempranas del desarrollo, haciendo necesario definir y modelar los procesos y las revisiones necesarias para realizar esto de la mejor manera posible.

Palabras Clave: *seguridad, estándar, política, vulnerabilidad, amenaza, modelo, pruebas, perfil UML, desarrollo.*

ABSTRACT

Security is a software engineering area which has not been relevant in the software development lifecycle until few years ago; in fact, it has been related to the testing phase at the end of the process. But this approach has changed and the security is becoming an important feature that can be added to the systems at the development phase although is not enough as security must be included at design. Keeping this in mind, this days is normal to think about security at earlier phases of the development process where it is necessary to define and modeling the process that are needed to do this in the best possible way.

Keywords: *security, standard, policy, vulnerability, threat, model, test, UML profile, development.*

TABLA DE CONTENIDOS

GLOSARIO

Credenciales	: permiten a un usuario identificarse ante un sistema.
Delivery	: es el proceso de desplegar una aplicación en un ambiente determinado.
ID	: identificación de usuario, puede ser el nombre de la persona, un número, etc.
Hackear	: entrar ilegalmente a un sistema, sin tener oficialmente los permisos necesarios para hacerlo.
Kernel	: núcleo central de una aplicación, es la base sobre la cual se ejecutan los procesos que realizan las diversas tareas hasta interactuar con el usuario.
Log	: archivo, generalmente de texto plano, que registra actividades de cualquier tipo.
Peer Review	: proceso de revisar un documento u otros artefacto con diferentes personas, las cuales emitirán su opinión para aprobarlo o realizar mejoras.
Release	: es una distribución de una versión, que no es la final, de un software.
Script	: es un listado de instrucciones que un proceso automático puede interpretar.
Tarjetahabiente	: persona cualquiera que es poseedora de una tarjeta, ya sea de crédito o débito.
Token	: dispositivo electrónico que permite agregar una mayor seguridad a una aplicación. Este genera una clave dinámica que solo la organización que lo emitió y el token mismo conocen.
Workflow	: permite automatizar y controlar la secuencia de acciones, actividades o tareas que se realizan para la ejecución de un proceso, incluyendo el seguimiento del estado de cada una de sus etapas.

1 INTRODUCCIÓN

En la actualidad cada vez se realizan una mayor cantidad de actividades a través de Internet o de otras redes, ya sean privadas o públicas, para las cuales se utilizan datos que son sensibles y que si son accesibles por otras personas pueden comprometer la privacidad y el patrimonio de personas o empresas. Este tipo de aplicaciones, generalmente, ofrecen servicios de compra de productos o pago de servicios, por lo que una falla en las medidas de seguridad de los datos puede provocar un gran daño desde el punto de vista económico.

Debido a este cambio en las maneras en que se utilizan las redes, se hace necesario verificar que las aplicaciones sean capaces de resguardar la privacidad de los usuarios a lo largo de todo el proceso de interacción. Para esto se comenzó a contratar a los mismo *hackers* o atacantes de los sistemas para que revelaran o encontraran las vulnerabilidades que permitieron el acceso indebido o el robo de información.

Mientras los sistemas y las medidas de seguridad necesarias no tuvieron una complejidad considerablemente alta, esta forma de verificar la seguridad fue bastante útil, pero con el tiempo se fue haciendo necesaria una metodología de pruebas de seguridad que los *hackers* no tenían, ya que generalmente estos no tenían una preparación formal en estos temas.

A mayor complejidad de los sistemas, se necesita una mejor planificación de las pruebas, lo que era muy difícil sin un proceso formal y establecido. De esta manera comenzaron a surgir una serie de estándares de seguridad que las aplicaciones y organizaciones debían cumplir, lo cual se verificaba una vez que las aplicaciones estaban terminadas. El problema de esto es que si se encuentra un error en el diseño hay que realizar un gran esfuerzo en retrabajo, por lo que se comenzó a hacer necesaria una manera de integrar la seguridad en el proceso de desarrollo formal, de manera que esta fuera considerada como un requerimiento más.

Existen una serie de esfuerzos en este sentido, llegando a hablar de ciclo de desarrollo seguro, pero sin una mayor definición en cuanto a los procesos y las tareas cotidianas que se deben realizar. Trabajos más específicos en este sentido han llegado a trabajar sobre perfiles UML, los cuales no son más que extensiones y personalizaciones del tradicional lenguaje de modelado.

Un perfil UML permite especificar un lenguaje tan general como UML, logrando acercarlo a dominios específicos o para considerar otro tipo de requerimientos, diferentes de los funcionales, en los mismos diagramas. De esta manera se ha logrado integrar el lenguaje proveniente del dominio de la seguridad informática en las etapas del proceso de desarrollo en que antes no se tenían en cuenta estos aspectos.

Aún así, queda camino por recorrer, ya que no existe un perfil UML estándar en este sentido y los que existen, aún presentan sus desventajas o están enfocados a aspectos demasiados específicos de la seguridad. Por ejemplo, uno de los perfiles que más se destaca, UMLSec, no es capaz de diferenciar entre usuarios conocidos o desconocidos para un sistema, aspecto que es de gran relevancia, ya que no se puede tratar de la misma manera a

estos usuarios, tanto desde el punto de vista de los accesos, como de las validaciones que se deben hacer a los datos que ingresan o que se les muestra por pantalla.

En el presente documento se espera proponer un perfil UML que ayude al desarrollo de sistemas seguros y que agregue puntos de verificación previos a las pruebas funcionales con el fin de evitar que se generen errores de este tipo al momento de codificar. Se comienza con una descripción de la metodología y planificación del trabajo, considerando los objetivos generales y específicos. Se continúa con una definición del concepto de seguridad, detallando los conceptos y modelos utilizados. En los capítulos posteriores se detallan las políticas y estándares de seguridad utilizados por muchas organizaciones que están comenzando a darle importancia a este aspecto del desarrollo de aplicaciones, para continuar con una descripción sobre que son los perfiles UML y como ayudan a considerar nuevos conceptos

Finalmente se detallará la propuesta de perfil UML, con los productos o resultados que se deben esperar de este. Además, se definirá el problema con el que se verificará lo propuesto, detallando los requerimientos que deberán ser modelados, para luego plantear una solución a este, utilizando la propuesta de perfil UML que se presentó.

1.1 OBJETIVOS

El objetivo principal de la presente tesis es, en base al lenguaje de modelado UML, definir una especificación de este que permita incluir requerimientos y conceptos de seguridad informática al análisis y diseño de software. Esto con el fin de evitar que los problemas de seguridad a los que se ven enfrentados hoy en día las aplicaciones no sean descubiertos al momento de las pruebas o, peor aún, una vez que el sistema ya es productivo y se han sufrido ataques de usuarios malintencionados o hackers profesionales.

Para cumplir con esto se definieron una serie de objetivos intermedios o específicos, los cuales permitirán contextualizar los términos o conceptos que se utilizarán.

1.1.1 OBJETIVO GENERAL

El objetivo general de la presente tesis es:

- Definir un perfil UML que permita modelar los requerimientos de seguridad desde las etapas iniciales del proceso de desarrollo de software.

1.1.2 OBJETIVOS ESPECÍFICOS

Los objetivos específicos del proyecto son:

- Establecer el estado del arte de las pruebas de seguridad y de cómo se integran con el desarrollo de software.

- Definir la propuesta de perfil UML para desarrollo seguro y establecer los productos o artefactos que se obtendrán.

- Comprobar la utilidad de lo propuesto con su aplicación en el análisis y diseño de un caso de estudio.

1.2 PLAN DE TRABAJO

1.2.1 METODOLOGÍA

Este proyecto se centrará en un comienzo en la descripción del estado actual en que se encuentran los tópicos señalados en los objetivos, estableciendo sus características desde lo más general a lo más específico y relacionándolos con la idea principal de este tema, orientar todos los conocimientos adquiridos hacia la integración de las pruebas de seguridad al ciclo de desarrollo.

Una segunda etapa comenzará una vez que se tengan claras las tareas involucradas con sus correspondientes tipos de prueba, comenzando a establecer los procesos que se utilizarán para el ciclo de desarrollo.

Se terminará con un ejemplo práctico basado en el diseño de una pequeña aplicación que permita utilizar los procesos diseñados.

1.2.2 PLANIFICACIÓN

A continuación se presentan las actividades que se realizaron durante el desarrollo de la presente tesis:

- Se comenzó con una investigación del estado del arte, detallando el estado actual de las pruebas de seguridad y en qué consisten, complementado con algunos estándares y metodologías de pruebas.

- Establecimiento de la importancia de las pruebas de seguridad y porque deben ser integradas al proceso de desarrollo.

- Investigación sobre los diferentes tipos de pruebas de seguridad que se pueden realizar sobre una aplicación.

- Investigación sobre los perfiles UML y las aplicaciones en seguridad que existen.

- Definición del problema que servirá como ejemplo de aplicabilidad de la propuesta.

- Integrar las pruebas de seguridad al desarrollo de software, definiendo un perfil UML que incluya conceptos de seguridad.

- Definir los productos que deben ser generados en cada etapa del perfil propuesto, estableciendo el contenido y la estructura que deben tener.
- Comprobar la validez de lo propuesto modelando una solución el problema escogido aplicando el perfil UML propuesto.

2 SEGURIDAD

Se habla de seguridad en muchos aspectos de la vida diaria, principalmente se refiere a la sensación de sentirse seguro, de saber que nada malo ocurrirá. La seguridad se puede verificar en los siguientes aspectos:

- Informática
  - Seguridad de datos.
  - Seguridad de redes.
  - Seguridad de aplicaciones.
- Física
  - Control de accesos.
  - Seguridad doméstica.

En general, la seguridad física es responsabilidad de entidades u organizaciones especializadas en este tema, ya que involucra procedimientos y experiencia que no se aprenden por si solos, sino que son enseñados en academias dedicadas a esto. Además existen convenciones o costumbres que regulan o recomiendan ciertos comportamientos que permiten mantenerse seguro o "fuera de peligro" a las personas.

Lo anterior es similar a la seguridad informática, ya que si bien existen empresas u organizaciones que se dedican a velar por la seguridad, ya sea generando recomendaciones o vendiendo un servicio, existen una serie de convenciones o costumbres que permiten mantenerse alejado de los peligros o por lo menos hacerlos menos probables.

2.1 CONCEPTOS DE SEGURIDAD

Existen una serie de conceptos clave en el vocabulario de seguridad informática que serán utilizados constantemente a lo largo de este documento, a continuación se presenta una definición de cada uno de ellos [TrustWave, 2008].

a. **Riesgo:** es una función de la probabilidad de que una determinada fuente de amenaza utilice una vulnerabilidad potencial en particular, y el impacto resultante de tal evento adverso para la organización [NIST, 2002].

b. **Vulnerabilidad:** es una falla o debilidad en los procedimientos, diseño, implementación o controles internos de la seguridad de sistemas, la que puede ser utilizada resultando en una brecha de seguridad o una violación de las políticas de seguridad [NIST, 2002].
c. **Amenaza:** es el potencial de una fuente de amenazas en particular para poder explotar exitosamente una vulnerabilidad [NIST, 2002] y que puede causar un impacto indeseado en alguna aplicación.
d. **Autenticación:** es el proceso de verificar una identidad, esto se puede hacer de tres maneras, las cuales no son excluyentes:
    - **Algo que se es:** la autenticación se realiza por medio de una propiedad física del individuo, por ejemplo
        - Biometría, como el escaneo de retina.
        - Huella digital.
        - Geometría de la palma de la mano.
        - Reconocimiento facial.
    - **Algo se tiene:** la autenticación se realiza con algo que posee el individuo, por ejemplo:
        - Código de barras.
        - RFID.
        - Cable Wiegand.
        - Clave que expira periódicamente.
    - **Algo se sabe:** la autenticación se realiza con algo que solo debe saber el individuo, por ejemplo: **c**lave o password.

Para una mayor seguridad se puede utilizar una combinación de los tres tipos de autenticación, tal como se utiliza en la actualidad en algunos bancos, los cuales utilizan algo que se sabe más algo que se tiene.
El proceso de autenticación es uno de los puntos más susceptibles de ataque, ya que si un atacante es capaz de ganar acceso a una aplicación puede simular ser otra persona con diversos accesos a información, la cual será posible obtener ilícitamente. Estos ataques pueden ser realizados a través de, por ejemplo [TrustWave, 2008]:

- **Inyección SQL:** consiste en la inserción de una consulta SQL a través de los datos de entrada desde el cliente hacia la aplicación. Un ataque de inyección de SQL exitoso puede leer datos sensibles desde una base de datos, modificarla, ejecutar operaciones de administración, etc.

- **Enumeración:** consiste en la utilización de listados de datos, los cuales son utilizados uno a uno para intentar acceder a un sistema u obtener un determinado resultado. Estos también son conocidos como ataques de fuerza bruta.

- **Reproducción:** a través de código insertado maliciosamente en un determinado sitio se obtiene la información de identificación de un usuario (sesiones, cookies, etc) para luego utilizarlas simulando ser este.

2.1.1 SUPERFICIE DE ATAQUE

Las aplicaciones pueden, eventualmente, tener una serie de interfaces, tanto hacia otros sistemas como hacia los usuarios, los cuales pueden ser de diversos tipos y tener diversos niveles de acceso.

Con superficie de ataque nos referimos a áreas de la aplicación que puede ser atacadas o contener vulnerabilidades. Aunque estas pueden no necesariamente contener vulnerabilidades son, de todas maneras, posibilidades que un atacante puede utilizar. Mientras más áreas sean accesibles mayor será la superficie de ataque.

La superficie de ataque puede hacer referencia no sólo a interfaces, sino que también a problemas que pueden existir a nivel de código, como el "código muerto", el cual no es más que trozos de código que ya no están operativos. Esto puede ser debido a, por ejemplo:

- Funciones o métodos que ya no se utilizan, porque fueron reemplazados.

- Trozos de código que nunca se ejecutan, ya que por un cambio en la implementación no se cumple la condición necesaria para que se ejecuten.

Es posible disminuir o mantener controlada la superficie de ataque a través de buenas prácticas al momento de diseñar y codificar un sistema, como las siguientes:

- **Principio del menor privilegio:** se asignan los privilegios exactos que necesita un usuario para realizar sus actividades, esto minimiza los riesgos del sistema ante robos de credenciales, ya que un atacante sólo podría realizar acciones que están permitidas a su víctima.

- **Funcionalidad mínima requerida:** implementar lo que los requerimientos y el diseño indican, sin invertir tiempo ni esfuerzo en ideas que surjan en el momento, ya que al no estar consideradas dentro del diseño original pueden producir vulnerabilidades.

- **Minimizar las entradas controladas por el usuario:** al acotar las posibilidades de entradas libres de datos para el usuario, se minimizan los riesgos de posibles ataques, ya que son estas las vías que utilizan los atacantes para llegar a la aplicación.

- **Aplicación apropiada de filosofías de validación de entradas:** se debe definir un estándar para las validaciones de las entradas de usuario, a modo de un mínimo deseable para toda la aplicación, con el fin de que no existan puntos débiles que permitan aprovechar vulnerabilidades que puedan existir. Estas validaciones deben bloquear los caracteres que usualmente se utilizan, como las comillas, los símbolos <>, etc.

2.2 MODELOS DE SEGURIDAD

Existen diversos modelos de seguridad, que se pueden aplicar dependiendo del dominio en que se necesiten. Los más comunes son lo que se describen a continuación [TrustWave, 2008].

2.2.1 MODELO DE CEBOLLA

Es una metáfora para las defensas por capas, un ejemplo físico serían los castillos medievales, los cuales contaban con:

- Un foso.

- Muros exteriores con torres.

- Guardias del Castillo.

- Habitaciones del rey.

El objetivo es mantener a salvo al rey y a la ciudad, por lo que si una defensa fallaba, la otra aún estaba en pie. En este sentido la seguridad informática no es diferente, ya que las aplicaciones deben ser seguras desde distintos ángulos:

- **Seguridad Física:** es la seguridad de las instalaciones que soportan a la aplicación que está siendo utilizada de forma que no puedan ingresar personas no autorizadas. Poe ejemplo: el acceso a la sala de servidores, instalaciones de los equipos de desarrollo, oficinas importantes, etc.

- **Seguridad en la infraestructura:** se refiere a la seguridad de las redes y las comunicaciones, a los mecanismos de respaldo y de contingencia, etc. En general es la seguridad de la infraestructura tecnológica que provee los servicios que permiten operar a una determinada aplicación. Además hace

referencia a las configuraciones de los servidores, de forma de corregir las vulnerabilidades conocidas y esconder información útil para los atacantes.

- **Configuración de las aplicaciones:** las aplicaciones desarrolladas deben estar configuradas correctamente para no inducir vulnerabilidades en la infraestructura y no revelar información importante al, por ejemplo, producirse errores durante su ejecución.

- **Validación de entradas de usuario:** toda entrada de datos por parte de los usuarios debe ser tratada como un potencial ataque, por lo que estos deben ser validados, filtrados, truncados, etc. Esto se puede realizar de dos formas, prohibiendo el ingreso de ciertos caracteres prohibidos o filtrando posteriormente lo que se haya ingresado.

- **Mecanismos de autenticación:** se deben proveer formas de identificación de los usuarios acorde a la importancia de los datos almacenados, esto se puede realizar a través de claves, características biométricas, etc. Se puede utilizar cualquier método siempre que este sea capaz de identificar únicamente a un usuario en particular.

- **Manejo y recuperación de errores:** se debe administrar la forma en que se despliegan los errores hacia el usuario, ya que de no realizarse es posible que se revelen datos importantes sobre la infraestructura, como: servidor utilizado, versión, etc. Esta información permite utilizar las vulnerabilidades conocidas que puedan existir para cada una de estas.

- **Generación de Log:** se deben mantener log para el registro de accesos y actividad importante por parte de los usuarios, de forma de facilitar posibles investigaciones posteriores a un ataque o filtración de información.

- **Seguridad de la base de datos y almacenamiento de datos:** deben existir mecanismos de encriptamiento y enmascaramiento acordes a la importancia de los datos almacenados. Esto para evitar que se puedan leer datos sensibles de los usuarios a pesar de un posible acceso no permitido a estos. Datos como: números de tarjetas de crédito, claves de usuario, etc. deben estar siempre encriptados o enmascarados.

La cantidad de funcionalidades accesibles que tiene un atacante sobre una aplicación se llama superficie de ataque. Mientras más grande sea ésta, más posibilidades tendrá el atacante.

2.2.2 TRÍADA ACI

ACI es la sigla en inglés para disponibilidad, confidencialidad e integridad.

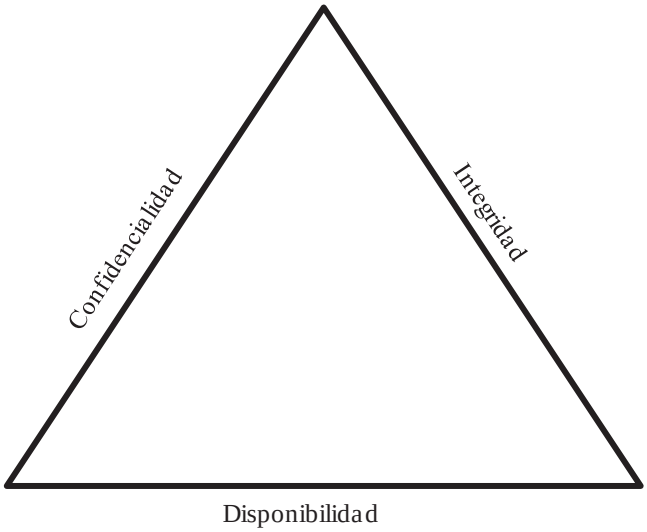

Figura 2.1 Modelo de seguridad ACI [TrustWave, 2008].

Se refiere a los aspectos que son importantes en un sistema informático, propone que con estos conceptos se está ofreciendo un nivel considerable de seguridad.

- **Disponibilidad (Availability):** se refiere a garantizar el acceso a los clientes, este aspecto es afectado por ataques DOS (sigla en inglés para Denegación de Servicio). Es importante, ya que si es descuidada se pierde el acceso por parte de los clientes a los productos o servicios que se estén ofreciendo.

- **Confidencialidad (Confidentiality):** se refiere a mantener resguardada la información sensible de la empresa, es decir, protegida de posibles ataques. El no tener en cuenta este aspecto permitirá a los atacantes tomar posesión de información valiosa que puede generar cuantiosas pérdidas para cualquier organización.

- **Integridad (Integrity):** se refiere a la usabilidad de los datos o del código de las aplicaciones. De no tener en cuenta este aspecto se puede tener una mala calidad de los datos organizacionales, disminuyendo la posibilidad de sacar conclusiones en base a estos. En cuanto al código, el descuido de la integridad produce que las distintas aplicaciones no sean capaces de interactuar correctamente, pudiendo provocar posibles brechas de seguridad.

2.2.3 PARKERIAN HEXAD

Es similar en cuanto a los conceptos a la triada ACI, extendiéndolo para una mayor inclusión. Aumenta el nivel de detalle para una definición más fácil de las amenazas [Parker, 1998].

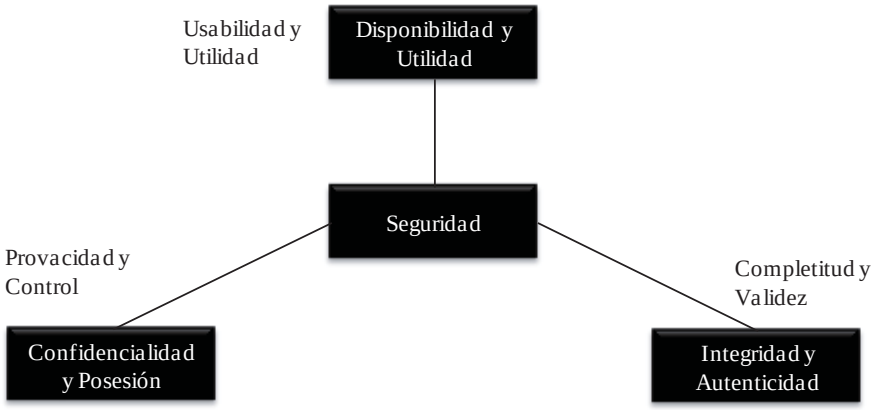

Figura 2.2 Diagrama de modelo Parkerian Hexad.

Se incluyen tres nuevos conceptos que agregan un mayor detalle:

- **Utilidad:** se refiere a utilizar elementos en las aplicaciones que realmente prestan una utilidad práctica para los objetivos planteados.
- **Posesión:** se refiere a la seguridad de tener la posesión de la información que es privada para una persona y que nadie más la puede tener.
- **Autenticidad:** se refiere a que nadie puede simular otra persona u organización, esto es sin obtener las credenciales de alguien más, sino que alterando de tal forma un documento o archivo de manera que se puede atribuir a alguien más.

2.3 APLICACIÓN DE MEDIDAS DE SEGURIDAD

La seguridad puede ser aplicada de distintas maneras y en distintas etapas del ciclo de desarrollo, así como también en las distintas capas que conforman la estructura de una aplicación.

2.3.1 SEGURIDAD EN LA ARQUITECTURA

Se refiere a los artefactos y elementos diseñados que describen donde los componentes de seguridad están posicionados y cómo interactúan con el resto de la arquitectura del sistema. Estos controles sirven para mantener los

atributos de calidad del sistema como la confidencialidad, integridad, disponibilidad, responsabilidad y aseguramiento [OSA, 2007].

2.3.2 SEGURIDAD EN SISTEMA OPERATIVO

A este nivel se diseñan sistemas operativos capaces de proveer servicios con un estándar altamente exigente de seguridad, como es el caso de las organizaciones militares, agencias de investigación, empresas con información sensible o estratégica, etc. Estos implementan procesadores especializados, así como procesos de administración de memoria, todo sobre un *kernel* diseñado específicamente.

2.3.3 SEGURIDAD EN LA CODIFICACIÓN

Cuando los sistemas operativos no son capaces de brindar un entorno seguro a las aplicaciones que son ejecutadas sobre él, estas deben ser capaces de resistir ataques y auto protegerse. Para esto deben ser codificadas siguiendo buenas prácticas de seguridad para evitar los ataques ya conocidos como buffer *overflow*, inyección de código, etc., además de prevenir los que puedan surgir. También se debe efectuar un correcto manejo de errores, con el fin de no revelar datos privados o detalles de la aplicación.

2.3.4 SEGURIDAD POR DISEÑO

Existen algunas buenas prácticas que se aplican a nivel de diseño de la aplicación que permiten mantener un nivel aceptable de seguridad, disminuyendo la superficie de ataque [TrustWave, 2008], por ejemplo:

- **Defensa en profundidad:** proteger en capas y con diferentes métodos con el fin de dificultar la penetración al sistema que se está protegiendo, además se utiliza para ganar tiempo demorando el ataque.

- **Principio del menor privilegio:** otorgar a un usuario, ya sea del tipo persona o sistema, los privilegios exactos que necesita, ya que si las credenciales de cualquiera de estos usuarios son obtenidas no podrá ganar más accesos de los que este tiene. Por ejemplo: no otorgar privilegios de administrador a ningún usuario adicional o que ingrese desde fuera de la aplicación, ya que si la credencial es robada un atacante puede hacer lo que desee.

- **Funcionalidades mínimas requeridas:** implementar los requerimientos exactos que se solicitaron, sin "adornar" la aplicación con funcionalidades extras que eventualmente pueden comprometer la seguridad.

- **Mínimas entradas controladas por el usuario:** entregar el control al usuario de las mínimas entradas de datos posibles, ya que cualquier entrada de datos es una alternativa que se le entrega al atacante.

- **Aplicar principios apropiados de validación de entradas de usuario:** existen tres formas de validar las entradas de usuario, listadas desde la más segura a la menos insegura:

  - **Valores Exactos:** tiene un listado con los valores permitidos.

  - **Lista Blanca:** maneja un listado con los patrones o formatos permitidos.

  - **Lista Negra:** maneja un listado con los patrones o formatos no permitidos.

2.4 VULNERABILIDADES MÁS COMUNES

En informática existen una serie de peligros o vulnerabilidades que pueden afectar de distinta manera a un sistema u organización. Estas van cambiando en el tiempo debido a que se corrigen los problemas que las hacían posibles o a que aparecen nuevas, para mantener un registro de las vulnerabilidades más importantes y peligrosas, la Fundación OWASP publica un reporte anual con las 10 más importantes del año, siendo el año 2007 las siguientes [OWASP, 2007]:

1. **Scripts a través de sitios (Cross Site Scripting - XSS):** permite ejecutar script en el browser de la víctima, con lo que se pueden obtener sesiones, introducir gusanos, etc.
2. **Defectos de inyección:** el más común es la inyección SQL, el cual introduce comandos SQL entre los datos que ingresa el usuario, pudiendo obtener datos privados.
3. **Ejecución de Código malicioso:** permite ejecutar código malicioso en un servidor, comprometiendo su integridad.
4. **Referencia insegura directa a objetos:** esto ocurre cuando un desarrollador expone una referencia directa a un objeto interno de la aplicación, ya sea un directorio, archivo, registro de base de datos o claves, como una URL o parámetro en un formulario.
5. **Falsificación de peticiones a través de sitios (Cross Site Request Forgery):** este ataque fuerza a un browser ya registrado de una víctima a enviar una petición pre-autenticada a la aplicación Web vulnerable, luego se fuerza al browser realizar una acción a favor del atacante.
6. **Filtración de información y manejo erróneo de errores:** una aplicación puede entregar datos importantes de su funcionamiento interno o configuración a través de los mensajes de error que muestra, lo cual permite al atacante realizar ataques más serios.
7. **Quiebre de autenticación y manejo de sesiones:** estos ataques comprometen claves, llaves, o *tokens* de autenticación que permiten asumir otras identidades.

8. **Almacenamiento criptográfico inseguro:** raramente las aplicaciones Web utilizan funciones criptográficas para proteger los datos y las credenciales, los atacantes aprovechan esto para robar información sensible como las tarjetas de crédito.
9. **Comunicaciones inseguras:** usualmente las aplicaciones fallan al momento de encriptar el tráfico de red para proteger información sensible.
10. **Error al restringir accesos a URLs:** se aprovechan fallos de seguridad para ingresar con URLs directas a sitios privados en una aplicación.

Al administrar y mitigar las vulnerabilidades antes mencionadas, se podrán evitar o disminuir la probabilidad de ocurrencia de una serie de ataques, estos son [OWASP, 2007]:

- Phishing (engaño al usuario, simulando ser la página oficial de alguna organización).
- Violación de privacidad.
- Robo de identidad.
- Compromiso de sistemas, alteración o destrucción de datos.
- Perdida financiera.
- Perdida de reputación.

2.5 PROBLEMAS DE PERFORMANCE

Los ataques de denegación de servicio son un problema conocido en el entorno de la informática, los cuales son asociados a ataques que vulneran la seguridad de una aplicación web. Pero la forma de prevenir o de ayudar a soportar de mejor manera estos ataques es a través de pruebas de performance que permitan desarrollar sistemas con este aspecto en mente o, en último caso, utilizarlas para optimizar los sistemas ya existentes.

Una forma de abordar el aspecto de performance desde un principio es considerar, en la etapa de diseño de una aplicación, características como:

- Volúmenes de datos esperados.
- Cantidad de Usuarios que se espera utilicen el sistema
- Cantidad máxima de usuarios que se espera atender

Además, se debe asegurar que el servidor está configurado correctamente para manejar sobrecargas y prevenir ataques de denegación de servicio. Verificar que el servidor ha sido optimizado apropiadamente [OWASP, 2008].

3 CRIPTOGRAFÍA

Criptografía es el estudio de las técnicas matemáticas relacionadas con aspectos de la seguridad informática tales como confidencialidad, integridad, autenticación de las entidades y autenticación de orígenes de datos. Esta no es lo único que importa al momento de proveer seguridad informática, pero sí cubre una serie de aspectos de ella.

Existen una serie de herramientas criptográficas, las cuales se muestran en la Figura 3.1 [Menezes, et al., 1996].

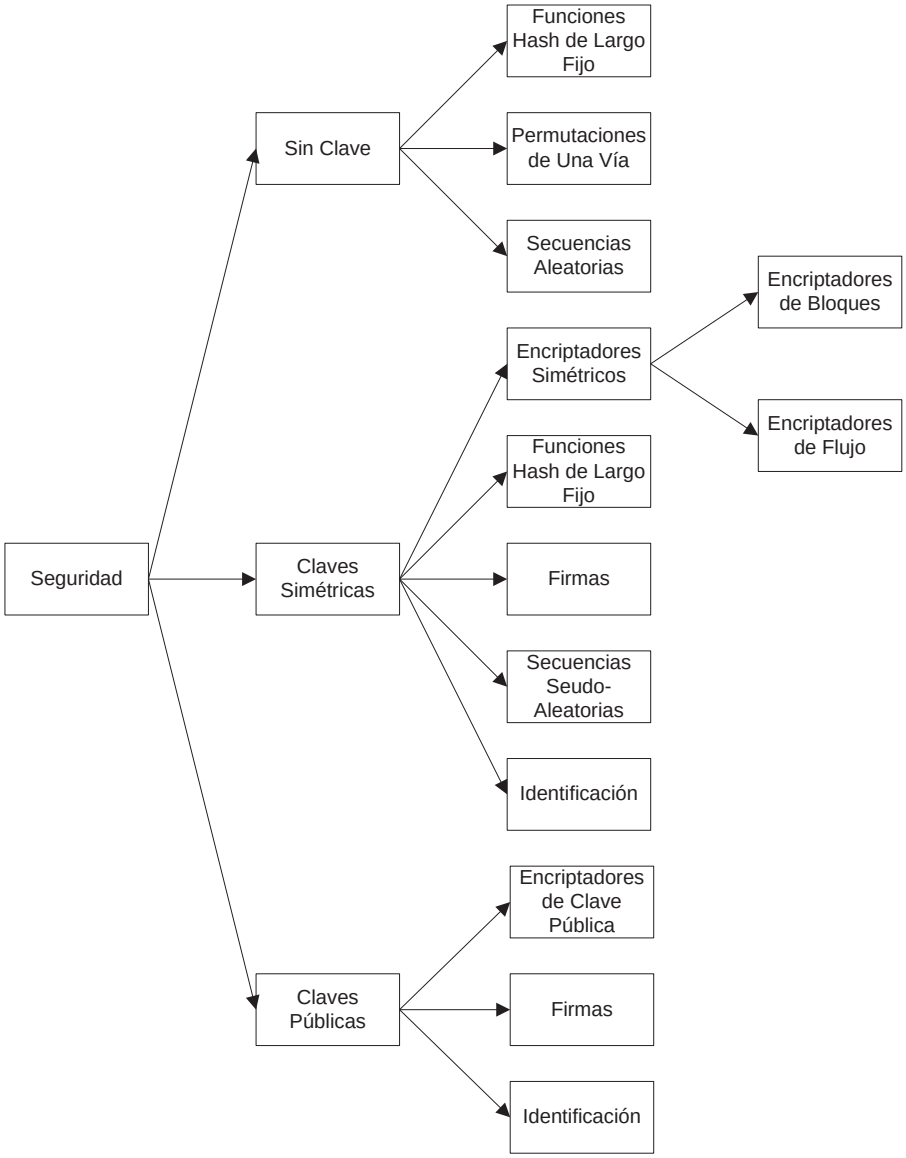

Figura 3.1 Clasificación de Sistemas Criptográficos.

3.1 SEGURIDAD SIN UTILIZACIÓN DE CLAVES

Se refiere a métodos de enmascaramiento y transformación de cadenas de caracteres que se realizan sin la necesidad de utilizar una clave. Estos son sólo de una vía, por lo que no es posible deshacer lo que ya ha sido transformado.

3.1.1 FUNCIONES HASH DE LARGO FIJO

Una función hash es una función computacionalmente eficiente que mapea cadenas de caracteres binarios de un largo dado hacia cadenas binarias de un largo fijo [Menezes, et al., 1996].

Para poder ser utilizada criptográficamente una función hash debe permitir que:

- Sea computacionalmente improbable encontrar dos valores que entreguen el mismo resultado.
- Sea computacionalmente improbable calcular a partir del hash el valor de entrada que lo produjo.

Las aplicaciones más comunes para las funciones hash son:

- Firmas Digitales: un mensaje es pasado por una función hash y luego el resultado es firmado digitalmente. El que recibe, aplica la misma función hash al mensaje y lo compara con lo que recibió firmado, si el hash es igual, el mensaje no ha sido alterado. Con esto es posible validar que el mensaje que se ha recibido es el original, sin tener que firma todo el mensaje completo, sino que solo el hash, ahorrando tiempo de procesamiento y complejidad adicional para mensajes largos.
- Integridad de Datos: en un instante de tiempo se obtiene el hash de un dato específico (almacenando el hash de forma segura), luego en el futuro se aplica la misma función hash y si el resultado es el mismo, el valor no ha cambiado.
- Almacenamiento de Claves: almacenar las claves en “claro” en una base de datos es peligroso, ya que eventualmente cualquier persona podría acceder a ella y obtener las claves de todos los usuarios. Para evitar esto, lo que se almacenan son los hash de las claves y al momento de autenticarse un usuario, se realiza el hash de la clave que este ingresó y se compara con el que está almacenado, si son iguales, se autoriza el ingreso del usuario.

3.1.1.1 FUNCIONES HASH MÁS UTILIZADAS

Algunas de las funciones hash más utilizadas son:

- **MD5:** este algoritmo surgió como una mejora al ya obsoleto MD4. Da como resultado cadenas de 128 bits, generalmente representadas como cadenas de 32 bits hexadecimales.

Su utilización está disminuyendo notablemente debido a que ya existen ataques bien documentados [Wang, et al., 2008], los cuales permiten encontrar colisiones. Se habla de colisión cuando se encuentran dos cadenas o archivos distintos que producen el mismo hash como resultado.

- **SHA:** para este algoritmo existen 4 implementaciones: SHA-1, SHA-256, SHA-384 y SHA512. Cuando un mensaje de largo menor a 2^{64} bits (para SHA-1 y SHA-256) o menor a 2^{128} bits (para SHA-384 y SHA-512) es procesado por el algoritmo, el resultado, llamado resumen del mensaje, es de largo entre 160 y 512 bits, dependiendo del algoritmo [FIPS 180-2, 2002].

  Para la variante SHA-1 ya se han encontrado colisiones, por lo que es recomendable utilizar al menos SHA-256, aunque ya se está haciendo necesario comenzar a pensar en un reemplazo del algoritmo SHA [Schneier].

3.2 CRIPTOGRAFÍA DE CLAVE SIMÉTRICA

En este tipo de criptografía la misma clave es utilizada tanto para encriptar como para desencriptar. Las ventajas de este tipo son:

- Es relativamente simple crear una clave fuerte.
- Las claves tienden a ser pequeñas en relación con la protección que entregan.
- Los algoritmos son relativamente económicos para procesar.

Una de las desventajas de este tipo de encriptación es que ambas partes deben conocer la clave con la cual se cifró el mensaje enviado, por lo que esta debe ser difundida para las personas involucradas en el intercambio. Esta difusión de la clave es la parte más débil del sistema, ya que ésta podría ser realizada por canales inseguros.

En la Figura 3.2 se describe el funcionamiento de la criptografía simétrica, en donde el mensaje encriptado viaja por un medio o canal inseguro, dejando en claro la utilización de una única clave para encriptar y desencriptar.

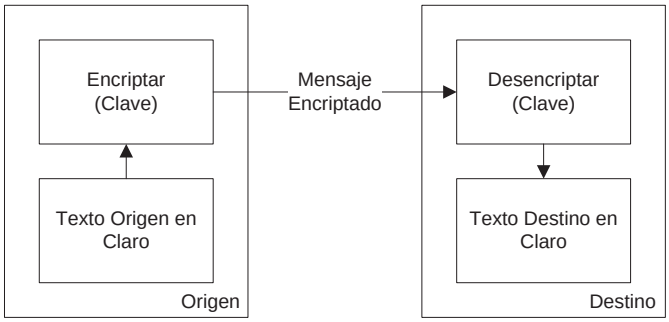

Figura 3.2 Esquema de funcionamiento de Clave Simétrica [Menezes, et al., 1996].

3.2.1 IMPLEMENTACIONES DE CLAVE SIMÉTRICA

Las siguientes son algunas implementaciones que utilizan el sistema de clave simétrica.

- **DES:** la clave de este algoritmo consiste de 64 bits binarios, de los cuales 56 son generados aleatoriamente y son utilizados por el proceso. Los otros 8 bits, los cuales no son usados, se pueden utilizar para detección de errores. Con los 8 bits de detección de errores se comprueba la paridad cada 8 bits (1 byte) [Daley, et al., 1999].

  Este algoritmo ya es posible quebrarlo por ataque de fuerza bruta, lo que fue demostrado en 1998, al utilizar tiempo de procesamiento disponible de una serie de computadores conectados a internet [EFF, 1998].

- **TripleDES:** este algoritmo utiliza el DES original, pero cifra utilizándolo 3 veces consecutivas. Se utiliza una clave de 192 bits, pero debido al diseño mismo del algoritmo la clave efectiva es de 112 bits [Daley, et al., 1999].

- **AES:** las entradas y salidas para este algoritmo son secuencias de 128 bits. La clave es una secuencia de 128, 192 o 256 bits, otros largos de entrada, salida o clave no son permitidos por el estándar [FIPS 197, 2001]. La unidad básica para este es 1 byte, una secuencia de 8 bits, tratados como una entidad. La entrada, salida y clave son divididas en grupos de 8 bits, formando arreglos de bytes.

  El 26 de noviembre de 2001 [FIPS 197, 2001], NIST anunció la publicación oficial del algoritmo AES (originalmente llamado Rijndael), convirtiéndose desde ese momento en el algoritmo estándar para el cifrado de datos, desplazando a DES y Triple DES.

- **Blowfish:** es un algoritmo de clave variable, esta puede ir de 32 a 448 bits. Este no cumple con todos los requerimientos para convertirse en un estándar, por lo que es recomendable utilizarlo en aplicaciones en donde la clave no cambie muy seguido, como comunicaciones o encriptadores automáticos de archivos. Blowfish es significativamente más rápido que DES, cuando es implementado en procesadores de 32 bits con una alta capacidad de memoria cache [Schneier, 1994].

3.3 CRIPTOGRAFÍA DE CLAVE PÚBLICA

La criptografía de clave pública soluciona la principal desventaja del sistema de clave simétrica, ya que utiliza un par de claves, una privada y una pública. Como su nombre lo dice, la clave pública puede estar disponible para todos lo que la necesiten, ya que es con esta clave que se encriptan los mensajes, de forma que sólo será posible desencriptarlos con la correspondiente clave privada. A su vez es posible firmar mensajes con la clave privada, de

forma que es posible saber de forma precisa quién lo envió a través de la correspondiente clave pública asociada . De esta manera, es posible enviar mensajes cifrados dirigidos a una persona en específico, sin la necesidad de tener que enviar la clave con la que realizó esto, como ocurre con las claves simétricas.

La principal desventaja de este sistema es el alto costo de procesamiento que produce [Menezes, et al., 1996], por lo que es más utilizado para cifrar mensajes más pequeños.

En la Figura 3.3 se describe el funcionamiento de la criptografía de clave pública, en donde el proceso comienza con la creación de las claves y la difusión de la clave pública. Posteriormente se cifra el mensaje con esta clave y se envía a través de un canal inseguro (al igual que la clave pública), para luego ser desencriptado con la clave privada del destinatario.

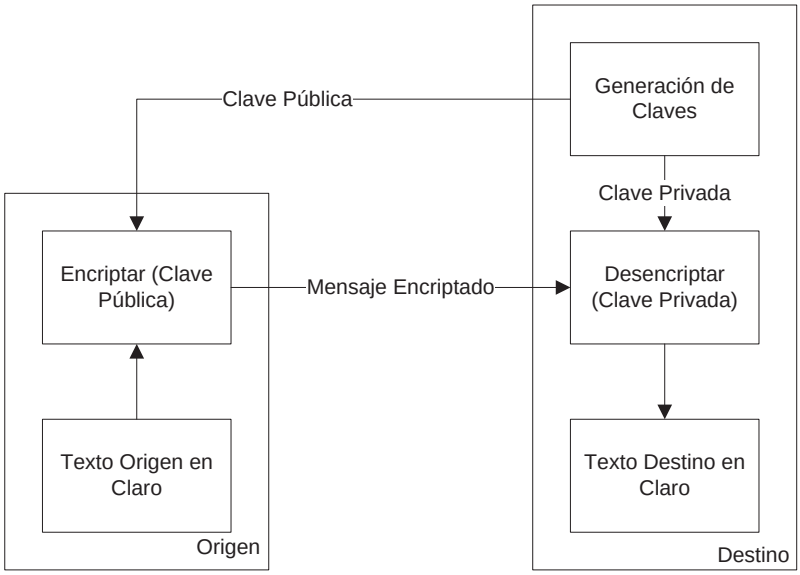

Figura 3.3 Esquema de funcionamiento de Clave Pública [Menezes, et al., 1996].

3.3.1 IMPLEMENTACIONES DE CLAVE PÚBLICA

El algoritmo RSA es la implementación más conocida de este tipo de criptografía [Menezes, et al., 1996]. Puede ser utilizado tanto para encriptar como para firmar digitalmente. Se basa en la utilización de números primos grandes para la encriptación, los cuales son costosos de encontrar y operar con ellos.

La implementación trabaja exactamente de la misma manera que la definición del tipo de criptografía de clave pública, ya que utiliza claves públicas y privadas, utilizando la primera para encriptar y la segunda para firmar.

RSA es un algoritmo considerablemente más lento que DES, por lo cual es principalmente utilizado para firmas digitales y enviar cifradas las claves de algoritmos simétricos.

3.3.2 TRANSPORT LAYER SECURITY (TLS)

TLS es el sucesor de SSL, sirviendo este último como base para su diseño. El objetivo principal de TLS es entregar privacidad e integridad de datos entre dos aplicaciones que se deban comunicar. El protocolo se compone de dos capas: el protocolo de registro y el protocolo de negociación.

El protocolo de registro ofrece seguridad en la conexión, la cual posee dos propiedades básicas:

- La conexión es privada: criptografía simétrica es utilizada para el cifrado de datos, en donde las claves generadas son únicas para cada conexión y están basadas en un código negociado por otro protocolo (como el protocolo de negociación TLS). El protocolo de registro también puede ser usado sin cifrado.

- La conexión es confiable: el transporte del mensaje incluye una revisión de integridad basado en un MAC codificado. Las funciones hash seguras son utilizadas para los cálculos de los MAC. El protocolo de registro puede operar sin MAC, pero generalmente sólo es utilizado en este modo mientras otro protocolo está usándolo como transporte para negociar parámetros seguros.

El protocolo de registro de TLS es utilizado para encapsular una variedad de protocolos de alto nivel. En tales protocolos, el protocolo de negociación TLS, permiten a cliente y servidor autenticarse mutuamente y negociar el algoritmo de encriptación y las claves antes que el protocolo de la aplicación trasmita o recibe el primer byte de datos. El protocolo de negociación TLS provee seguridad en la conexión, la cual tiene 3 propiedades básicas:

- Las identidades de los involucrados pueden ser autenticadas utilizando criptografía asimétrica o de clave pública. Esta puede ser opcional, pero generalmente es requerida para al menos uno de los participantes.

- La negociación de las claves compartidas es segura: las claves negociadas no están disponibles para los curiosos, y para cualquier conexión autenticada la clave no puede ser obtenida, incluso para un atacante que está al medio de la conexión escuchando.

- La negociación es confiable: ningún atacante puede modificar la conversación de negociación sin ser detectado por los participantes de la conversación.

Una ventaja de TLS es que es independiente del protocolo de la aplicación. Protocolos de más alto nivel pueden ubicarse por encima de TLS de forma transparente. Sin embargo, TLS estándar no especifica como otros protocolos agregan seguridad con él, las decisiones sobre como iniciar la negociación y cómo interpretar los certificados de autenticación intercambiados son dejadas a juicio de los diseñadores e implementadores de los protocolos que corren sobre TLS [RFC5246, 2008].

4 POLÍTICAS DE SEGURIDAD

Las aplicaciones seguras no son desarrolladas de manera espontánea, sino que son el resultado de una serie de procesos que un equipo u organización deben seguir para producir el resultado esperado en cuando a la capacidad de resistir diversos tipos de ataque.

En general, para desarrollar una aplicación segura, los siguientes son los aspectos mínimos [OWASP, 2007]:

- La dirigencia de la organización con un compromiso por la seguridad.
- Políticas de seguridad definidas y escritas siguiendo los estándares nacionales.
- Una metodología de desarrollo con actividades y puntos de verificación de seguridad adecuados.
- Generación de *releases* de manera segura y administración de la configuración.

4.1 COMPROMISO ORGANIZACIONAL HACIA LA SEGURIDAD

Las organizaciones que tienen la seguridad incorporada desde los niveles más altos, generalmente, producirán aplicaciones que cumplen los principios básicos de la seguridad de la información [OWASP, 2007]. Este es el primer paso de muchos que llevarán desde desarrollar aplicaciones "posiblemente seguras" a "un poco seguras". Estas organizaciones revisan los riesgos contra su política de seguridad, haciendo posible saber que riesgo será aceptado, mitigado o asignado.

Las organizaciones que no tienen políticas de seguridad, no tienen forma de revisar estos riesgos, por lo tanto no es posible clasificar correctamente estos riesgos, ni tampoco tener los puntos de control necesarios para sus revisiones, los cuales pueden estar mal definidos.

Muchas organizaciones definen políticas de seguridad derivadas de la norma ISO 17799, aunque si se está en EE.UU. se puede utilizar COBIT, también ocasionalmente se utilizan ambos. No existe una regla o estándar para producir políticas de seguridad de la información, pero en general [OWASP, 2007]:

- Si se es públicamente transado en la mayoría de los países, se debe tener políticas de seguridad de la información.
- Si se es públicamente transado en los EEUU, se debe tener políticas de seguridad de la información que cumplan con los requerimientos Sarbanes-Oxley (SOX), lo cual generalmente significan controles COBIT (sigla en inglés de Objetivos de Control para la Información y Tecnologías relacionadas).
- Si se es privado, pero se tiene una cantidad importante de empleados, probablemente se necesite una.

Es correcto mezclar y ajustar controles provenientes de COBIT o ISO 17799 y casi cualquier otro estándar de seguridad de la información, ya que es difícil que se contradigan en los detalles.

4.1.1 COBIT

COBIT (sigla en inglés de Objetivos de Control para la Información y Tecnologías relacionadas) provee buenas prácticas a través de un marco de trabajo de dominio y proceso, que presenta actividades en una estructura lógica y administrable. Las buenas prácticas presentadas por COBIT representan el consenso de expertos.

Está fuertemente enfocado más en el control y menos en la ejecución. Estas prácticas ayudarán a optimizar las inversiones en TI, asegurar la entrega del servicio y proveer una medida contra la cual juzgar cuando las cosas van mal.

Este marco de trabajo está estructurado en torno a 4 dominios [IT Governance Institute, 2007]:

- **Planificación y organización:** este dominio cubre la estrategia y la táctica, ocupándose de la identificación de la manera en que TI puede contribuir mejor para alcanzar los objetivos del negocio. La realización de la visión estratégica necesita ser planificada, comunicada y administrada para diferentes perspectivas. Este dominio típicamente se enfoca en las siguientes preguntas de administración:
    - ¿Están las estrategias de TI y del negocio alineadas?
    - ¿Está la empresa alcanzando la utilización óptima de recursos?
    - ¿Entienden todos en la organización los objetivos de TI?
    - ¿Están siendo entendidos y administrados los riesgos de TI?
    - ¿Es la calidad de los sistemas TI la apropiada para las necesidades del negocio?
- **Adquisición e implementación:** para darse cuenta de la estrategia de TI, las soluciones necesitan ser identificadas, desarrolladas o adquiridas, así como implementadas e integradas al proceso de negocio. Además, los cambios y las mantenciones de los sistemas existentes están cubiertos por este dominio para asegurarse de que las soluciones continúan cumpliendo los objetivos del negocio. Este dominio típicamente se enfoca en las siguientes preguntas de administración:
    - ¿Están los nuevos proyectos cercanos a proveer soluciones que cumplen con las necesidades del negocio?
    - ¿Están los nuevos proyectos cercanos a ser entregados a tiempo y cumpliendo con el presupuesto?

- ¿Trabajarán los nuevos sistemas apropiadamente cuando sean implementados?
- ¿Serán los cambios realizados sin trastornar las operaciones actuales del negocio?

- ***Delivery* y soporte:** este dominio se ocupa de la entrega de los servicios requeridos. Los cuales incluyen la entrega del servicio, administración de la seguridad y continuidad, soporte del servicio para usuarios, y la administración de los datos e instalaciones operacionales. Este dominio típicamente se enfoca en las siguientes preguntas de administración:
  - ¿Están los servicios siendo entregados alineados con las prioridades del negocio?
  - ¿Están siendo los costos de TI optimizados?
  - ¿Es la fuerza de trabajo capaz de utilizar los sistemas TI en forma productiva y segura?
  - ¿Están la confidencialidad, integridad y disponibilidad adecuadamente ubicados para la seguridad de la información?

- **Monitoreo:** todos los procesos TI necesitan ser regularmente evaluados en el tiempo para asegurar su calidad y cumplimiento con los requerimientos de control. Este dominio evalúa administración del desempeño, monitoreo del control interno, regulaciones, cumplimiento y gobernación. Este dominio típicamente se enfoca en las siguientes preguntas de administración:
  - ¿Es el desempeño de TI medido para detectar problemas antes que sea demasiado tarde?
  - ¿Asegura la administración que los controles internos son efectivos y eficientes?
  - ¿Puede el desempeño de TI ser relacionado con los objetivos del negocio?
  - ¿Están los controles de confidencialidad, integridad y disponibilidad adecuadamente en su lugar para la seguridad de la información?

4.1.2 ISO 17799

Es un marco de trabajo para la administración de la seguridad de la información basado en riesgo derivado de los estándares AS/NZS 4444 y BS 7799. Es un estándar internacional utilizado fuertemente en muchas organizaciones fuera de EE.UU [ISO, 2005].

La seguridad de la información se define en el estándar como:

- **Preservación de la confidencialidad:** asegurar que sólo quienes estén autorizados pueden acceder a la información.

- **Integridad:** asegurar que la información y sus métodos de proceso son exactos y completos.
- **Disponibilidad:** asegurar que los usuarios autorizados tienen acceso a la información y a sus activos asociados cuando lo requieran.

4.1.3 SARBANES-OXLEY

Es una ley de Estados Unidos también conocida como el Acta de Reforma de la Contabilidad Pública de Empresas y de Protección al Inversionista. También es llamada SOX o SARBOX.

SOX nace en Estados Unidos con el fin de monitorear a las empresas que cotizan en bolsa, evitando que las acciones de las mismas sean alteradas de manera dudosa, mientras su valor sea menor. Su finalidad es evitar fraudes y riesgo de bancarrota, protegiendo al inversor.

Esta ley está formada por 11 apartados que describen mandatos y requerimientos específicos para el negocio financiero, los cuales contienen una serie de secciones:

1. **Consejo de revisión de contabilidad para compañías públicas:** contiene 9 secciones, en las cuales se establece el consejo de revisión de contabilidad para compañías públicas, de forma de proveer una revisión independiente de organizaciones de contabilidad pública proveyendo servicios de auditoría. También crea un consejo central de revisión, el cual trabaja con auditores registrados, definiendo procesos específicos y procedimientos para auditorías de cumplimiento, inspeccionado los controles de calidad y las políticas, y reforzando el cumplimiento de los mandatos específicos de SOX. Estas secciones son:
    - Sección 101: Establecimiento; disposiciones administrativas.
    - Sección 102: Registro ante la Junta.
    - Sección 103: Auditoria, control de calidad, y normas y reglamentos de independencia.
    - Sección 104: Inspecciones de empresas públicas de contabilidad registradas.
    - Sección 105: Investigaciones y procedimientos disciplinarios.
    - Sección 106: Empresas públicas de contabilidad extranjeras.
    - Sección 107: Comisión de Supervisión de la Junta.
    - Sección 108: Normas de contabilidad.
    - Sección 109: Financiamiento.

2. **Independencia del auditor:** consiste de 9 secciones, establece estándares para la independencia de los auditores externos, de forma de limitar los conflictos de intereses. Además establece nuevos requerimientos para la aprobación de los auditores, políticas de rotación de auditores, problemas de conflictos de intereses y requerimientos de reportes de los auditores. Por ejemplo, la sección 201 establece restricciones a las compañías, respecto de la posibilidad de hacer otros tipos de negocios aparte de auditar los mismos clientes. Estas secciones son:

    - Sección 201: Servicios fuera del alcance de la práctica de los auditores.
    - Sección 202: Requisitos pre-aprobatorios.
    - Sección 203: Rotación del socio de auditoría.
    - Sección 204: Informes del auditor al comité de auditoría.
    - Sección 205: Modificaciones acordadas.
    - Sección 206: Conflictos de interés.
    - Sección 207: Estudio de la rotación obligatoria de las empresas públicas de contabilidad registradas.
    - Sección 208: Autoridad de la Comisión.
    - Sección 209: Consideraciones por las autoridades estatales reguladoras apropiadas.

3. **Responsabilidad corporativa:** consiste de 8 secciones y ordena que los ejecutivos *senior* asuman una responsabilidad individual sobre la precisión y completitud de los reportes financieros corporativos. Define la interacción entre los auditores externos y los comités de auditoría corporativa, especificando la responsabilidad de los ejecutivos corporativos sobre la precisión y validez de los reportes financieros. Enumera los límites específicos sobre los comportamientos de los ejecutivos corporativos y describe las pérdidas específicas de beneficios y penas civiles por el no cumplimiento. Por ejemplo: la sección 302 implica que el directorio de la compañía (CEO, CFO) deben certificar y aprobar la integridad de los reportes financieros de la compañía trimestralmente para establecer la contabilidad. Estas secciones son:

    - Sección 301: Comité de auditoría de compañías públicas.
    - Sección 302: Responsabilidad corporativa sobre los reportes financieros.

- Sección 303: Influencia impropia sobre la conducta de los auditores.
- Sección 304: Confiscación de ciertos bonos y ganancias.
- Sección 305: Excepciones y penalidades del funcionario y director.
- Sección 306: Operaciones con información privilegiada durante períodos de detención.
- Sección 307: Reglamentos de responsabilidad profesional para abogados.
- Sección 308: Fondos razonables para inversionistas.

4. **Divulgaciones financieras mejoradas:** consiste de 9 secciones, donde se describe los requerimientos de los reportes mejorados para las transacciones financieras. Requiere controles internos para asegurar la precisión de los reportes y divulgaciones financieras, ordena que existan en estos controles auditorías y reportes. Estas secciones son:
   - Sección 401: Revelaciones en informes periódicos.
   - Sección 402: Previsiones mejoradas para conflictos de interés.
   - Sección 403: Revelaciones de transacciones que involucran a la gerencia y accionistas principales.
   - Sección 404: Evaluación de la gerencia sobre los controles internos.
   - Sección 405: Excepciones.
   - Sección 406: Código de ética para los funcionarios financieros *senior*.
   - Sección 407: Revelación del experto financiero del comité auditor.
   - Sección 408: Revisiones mejoradas de las divulgaciones periódicas de los emisores.
   - Sección 409: Divulgaciones en tiempo real de los emisores.

5. **Analizar los conflictos de intereses:** consiste de una única sección, la cual incluye medidas diseñadas para ayudar a restaurar la confianza del inversor en los reportes de los analistas de seguridad. Define los códigos de conducta para los analistas de seguridad y requiere la divulgación de los conflictos de intereses conocidos. Estas sección es:

    - Sección 501: Tratamiento de los analistas de valores por asociaciones de valores registradas y bolsas de valores nacionales.

6. **Recursos y autoridad de la comisión:** consiste de 4 secciones y define prácticas para restaurar la confianza del inversor en los analistas de seguridad. También define la autoridad del SEC para censurar o inhabilitar a los profesionales de la seguridad de la práctica y define las condiciones bajo las cuales una persona puede ser inhabilitada de la práctica como corredor, asesor o distribuidor. Estas secciones son:

    - Sección 601: Autorización de apropiaciones.
    - Sección 602: Apariencia y práctica antes de la Comisión.
    - Sección 603: Autoridad de la Corte Federal para imponer sanciones.
    - Sección 604: Calificaciones de personas asociadas a corredores y negociadores.

7. **Estudios y reportes:** contiene 5 secciones y está enfocada en conducir la investigación para reforzar las acciones contra las violaciones hechas por las compañías asociadas al SEC y los auditores. Los estudios y reportes incluyen los efectos de las consolidaciones de las contabilidades públicas, el rol de las agencias de *rankeo* crediticio en la operación de los mercados de seguridad, violaciones de seguridad y reforzamiento de acciones. Estas secciones son:

    - Sección 701: Estudio e informe GEAO sobre consolidación de empresas públicas de contabilidad.
    - Sección 702: Estudio e informe de la Comisión referente a las agencias de evaluación de créditos.
    - Sección 703: Estudio e informe sobre infractores e infracciones.
    - Sección 704: Estudio de acciones vigentes.
    - Sección 705: Estudio de bancos de inversión.

8. **Rendiciones de cuentas corporativas y de fraude criminal:** consiste de 7 secciones y también es nombrado como el "Acto de Fraude Corporativo y Criminal de 2002". Describe penalizaciones criminales específicas para el fraude por manipulación, destrucción o alteración de registros financieros u otras interferencias con investigaciones, mientras provee ciertas protecciones para los delatores. Estas secciones son:

    - Sección 801: Título corto.
    - Sección 802: Sanciones penales por la alteración de documentos.
    - Sección 803: Deudas no deducibles, si incurrieron en una violación de las leyes de fraude de valores.
    - Sección 804: Estatuto de limitaciones por fraude de valores.
    - Sección 805: Revisión de las Pautas de sentencia Federal para la obstrucción de la justicia y fraude penal.
    - Sección 806: Protección para empleados de compañías tranzadas públicamente que provean evidencia de fraude.
    - Sección 807: Sentencias para accionistas defraudadores de compañías tranzadas públicamente.

9. **Mejoramiento de la penalización de crímenes de cuello blanco:** consiste de 2 secciones. Esta es también llamada "Acto de Mejoras a la Penalización del Crimen de Cuello Blanco del 2002". Incrementa las penalizaciones criminales asociadas con los crímenes y conspiraciones de cuello blanco (delitos cometidos por empleados). Recomienda guía fuertes y específicamente agrega errores para certificar los reportes financieros como una ofensiva criminal. Estas secciones son:

    - Sección 901: Título corto.
    - Sección 902: Intentos y conspiraciones para cometer fraude.
    - Sección 903: Responsabilidad penal por fraude lógico y físico.
    - Sección 904: Responsabilidad penal por violaciones al "Employee Retirement Income Security Act" de 1974.
    - Sección 905: Modificación de las guías de sentencia referentes a ciertos delitos de "cuello blanco" (delitos cometidos por empleados).

- Sección 906: Responsabilidad corporativa sobre los informes financieros.

10. **Devolución de impuestos:** consiste de 1 sección, la cual establece que el CEO debe firmar la devolución de impuestos de la compañía. Esta sección es:
    - Sección 1001: Opinión del Senado respecto de la firma de las devoluciones de impuestos corporativas por los CEOs.
11. **Rendiciones de cuenta del fraude corporativo:** consiste de 7 secciones. La primera sección recomienda un nombre para este apartado: "Acto de Fraude Corporativo Contable de 2002". Identifica el fraude corporativo y registra la manipulación como ofensas criminales, agrupando esas ofensas con penas específicas. También revisa las guías para las sentencias y fortalece sus penas. Esto permite al SEC congelar temporalmente pagos grandes o inusuales. Estas secciones son:
    - Sección 1101: Título corto.
    - Sección 1102: Manipulación de un registro para impedir un procedimiento Oficial.
    - Sección 1103: Autoridad de bloqueo temporal para la Comisión Valores e Intercambio (SEC).
    - Sección 1104: Modificación a las Guías de Sentencia Federales.
    - Sección 1105: Autoridad de la Comisión para prohibir a las personas prestar servicios como funcionarios o directores.
    - Sección 1106: Aumentar la responsabilidad penal bajo el Acta del SEC de 1934.
    - Sección 1107: Represalias contra los informantes.

4.1.4 PCI-DSS

El "Estándar de Seguridad Datos de la Industria de Tarjetas de Pago" (PCI-DSS) es un conjunto de requerimientos que ayudan a mejorar la seguridad de los datos en los procesos de pago. Fue desarrollado por las marcas de pago fundadoras del "Consejo de Estándares de Seguridad de la Industria de Tarjetas de Pago" (PCI SSC), en las cuales se incluyen: American Express, Discover Financial Services, JCB International, Mastercard, y Visa. Esto con el fin de facilitar la adopción de medidas consistentes para la seguridad de los datos sobre una base global [PCI, 2008].

El estándar incluye requerimientos para la administración de la seguridad, políticas, procedimientos, arquitectura de la red, diseño de software y otras medidas de protección. Intenta ayudar a las organizaciones a proteger proactivamente los datos de los clientes.

El centro de este estándar es un grupo de principios, acompañado de requerimientos, sobre los cuales los elementos específicos del estándar de seguridad de datos (DSS) esta organizado:

- **Construir y mantener una red segura**

  Requerimiento 1: instalar y mantener una configuración del cortafuego que proteja los datos del tarjetahabiente.

  Requerimiento 2: no usar las claves de los sistemas ni otros parámetros de seguridad con los valores por defecto entregados por el proveedor.

- **Proteger los datos de los tarjetahabientes**

  Requerimiento 3: proteger los datos almacenados de los tarjetahabientes.

  Requerimiento 4: encriptar las transmisiones de datos de los tarjetahabientes a través de redes abiertas y públicas.

- **Mantener un programa de administración de vulnerabilidades**

  Requerimiento 5: utilizar un software de antivirus que se actualice regularmente.

  Requerimiento 6: desarrollar y mantener aplicaciones y sistemas seguros.

- **Implementar medidas fuertes de control de acceso**

  Requerimiento 7: restringir acceso a los datos de los tarjetahabientes como parte del negocio.

  Requerimiento 8: asignar una ID única a cada persona con acceso a un computador.

  Requerimiento 9: restringir el acceso físico a los datos de los tarjetahabientes.

- **Monitorear y probar regularmente las redes**

  Requerimiento 10: rastrear y monitorear todos los accesos a los recursos de las redes y los datos de los tarjetahabientes.

  Requerimiento 11: probar regularmente los sistemas y procesos de seguridad.

- **Mantener una política de seguridad de la información**

  Requerimiento 12: mantener una política de seguridad que establezca una seguridad de la información

4.1.5 METODOLOGÍA DE DESARROLLO

El desarrollo por si solo no es lo suficientemente estructurado para producir aplicaciones seguras. Las organizaciones que deseen producir código seguro siempre necesitan una metodología que permita lograr ese objetivo.

Al escoger una metodología hay que tener en mente en tamaño del equipo que trabajará en el desarrollo, ya que de esto dependerá la que será más conveniente, ya que los pequeños deben pensar en las que sean más livianas que no definan demasiados roles. En cambio, los equipos grandes deben considerar las que les permitan escalar en sus necesidades.

Los siguientes son los atributos a considerar en una metodología de desarrollo:

- Gran aceptación del diseño, pruebas y documentación.
- Lugares para ubicar controles de seguridad, como: análisis de riesgo de amenazas, *peer review*, revisiones de código, etc.
- Funciona para el tamaño y madurez de la organización.
- Tiene el potencial para disminuir la cantidad de errores y mejorar la productividad del desarrollador.

4.1.6 ESTÁNDARES DE CODIFICACIÓN

Las metodologías no son estándares de codificación, cada organización necesitará determinar que utilizar basado en sus prácticas comunes o simplemente utilizar las mejores prácticas conocidas.

Los siguientes son los artefactos a considerar:

- Guías para la arquitectura.
- Niveles de documentación mínimos requeridos.
- Requerimientos obligatorios para las pruebas.
- Niveles mínimos de comentarios en el código y el estilo preferido.
- Uso del manejo de excepciones.
- Uso de los flujos de bloques de control.

- Métodos de nombramiento preferidos para las variables, funciones, clases y tablas.
- La preferencia entre código mantenible y legible sobre código complejo.

Los estilos de identación y tabulación, desde la perspectiva de la seguridad, no importan demasiado, ya que estos pueden ser corregidos con herramientas automatizadas o simplemente utilizar estilos en el editor de código, por lo que no se debe dar demasiada importancia a este aspecto.

4.1.7 CONTROL DEL CÓDIGO FUENTE

La ingeniería de software de alto nivel necesita el uso regular de mejoras al proceso de codificación, además de regímenes de prueba asociados. Todo el código generado y las pruebas deben ser posibles de ser revertidos y versionados.

Esto puede ser realizado copiado los archivos y carpetas a un servidor de archivos, pero esto es realizado de mejor manera por herramientas de versionado de código fuente, como Subversión, CVS, SourceSafe o ClearCase.

Las pruebas deben ser incluidas en el versionado, ya que pruebas para versiones posteriores no coinciden con las pruebas necesarias para versiones anteriores. Es muy importante aplicar la prueba que corresponde a la versión que se está revisando.

5 PERFILES UML PARA DESARROLLAR SISTEMAS SEGUROS

La toma de requerimientos funcionales y su posterior diseño es un área del desarrollo de sistemas informáticos, a pesar de los problemas que aún existen, bastante madura en la actualidad. Se tienen distintas metodologías y lenguajes de modelado que permiten plasmar gráfica y textualmente los requerimientos provenientes de los usuarios finales. Esto mismo no ocurre con los requerimientos no funcionales, especialmente con los relacionados con la seguridad.

Para esta clase de requerimientos existen los mismos problemas y consideraciones a tener, que con los requerimientos funcionales, ya que, dependiendo del dominio de aplicación, una mala implementación de uno de ellos puede provocar errores funcionales, problemas de disponibilidad o, peor aún, serio compromiso de la integridad y la confidencialidad de los datos que maneje o almacene el sistema desarrollado. Es sabido por todas las personas ligadas al desarrollo de sistemas que mientras más tarde se encuentra un defecto funcional, más caro sale, en cuanto a los recursos necesarios para corregirlo. Esto mismo ocurre con los problemas de seguridad, ya que, en muchas ocasiones, un error de este tipo puede ser originado por un problema de diseño, más que por uno de implementación.

Por esta razón es necesario considerar seriamente los requerimientos de seguridad desde la etapa de análisis de un nuevo sistema, para evitar que problemas de diseño, produzcan vulnerabilidades que serán costosas de corregir y que pueden ser utilizadas por cualquier atacante con acceso a este.

5.1 PERFILES UML

5.1.1 UML

El lenguaje de modelado unificado (UML, por sus siglas en ingles) ayuda a especificar, visualizar y documentar modelos de sistemas de software, aunque se puede utilizar para modelar procesos de negocio y otros sistemas no necesariamente de software. A través de las herramientas basadas en este lenguaje, se pueden analizar y diseñar soluciones, representando los resultados por medio de los diagramas definidos para esto.

Una de las razones por las que UML se ha convertido en un lenguaje estándar de modelado es que este es independiente del lenguaje de programación que se utilice. También se debe a que la notación UML es un lenguaje no una metodología [IBM, 2003]. Esto es importante, ya que un lenguaje, al contrario de una metodología, puede fácilmente acomodarse a los requerimientos de una organización sin requerir cambios.

Se puede modelar cualquier tipo de aplicación, corriendo sobre cualquier combinación de hardware, sistema operativo, lenguaje de programación y red. Al ser construido a partir de conceptos fundamentales de la orientación a objetos, incluyendo clase y operación, es una alternativa natural para lenguajes orientados a objetos y entornos como

C++, Java y C#, pero se puede utilizar para modelar aplicaciones desarrolladas sobre lenguajes de otro tipo [OMG, 2005].

Debido a que UML no es una metodología, no se requieren productos formales de trabajo [IBM, 2003], pero si provee una variedad de tipos de diagramas que, cuando se utilizan con una metodología, aumentan la facilidad de entendimiento de la aplicación en desarrollo.

5.1.1.1 DIAGRAMAS

UML provee una serie de diagramas para representar la información, entre los más útiles están: diagrama de casos de uso, diagrama de clases, diagrama de secuencia, diagrama de estados, diagrama de actividades y diagrama de componentes.

5.1.1.1.1 Diagrama de Casos de Uso

Este diagrama ilustra una unidad de una funcionalidad entregada por el sistema. El propósito final de un caso de uso es ayudar al equipo de desarrollo a visualizar los requerimientos funcionales del sistema, incluyendo la relación de los actores con los procesos esenciales, así como las relaciones a través de los diferentes casos de uso.

Se utilizan en la etapa de análisis para detallar como se relacionan los diferentes actores del sistema con sus componentes y las relaciones entre estos últimos. Son una herramienta clave al momento de definir los requerimientos, ya que permiten al usuario final visualizar de una manera más gráfica lo que el equipo de desarrollo entiende del sistema.

En general los diagramas de casos de uso muestran grupos de casos de uso, tanto como todos los casos de un sistema completo, como una parte de un determinado grupo con una funcionalidad en común.

5.1.1.1.2 Diagrama de Clases

Este diagrama muestra como las diferentes entidades se relacionan entre ellas, es decir, muestra las estructuras estáticas del sistema [IBM, 2003]. Puede ser utilizado para mostrar las clases lógicas, los cuales son típicamente los tipos de cosas sobre las que se habla por parte de las personas de las áreas de negocio. Estos también pueden ser utilizados para mostrar las clases de implementación, las cuales son uno de los elementos con los que los programados usualmente trabajan.

Un diagrama de clases de implementación probablemente mostrará algunas de las mismas clases que el diagrama lógico, pero no tendrá los mismos atributos.

5.1.1.1.3 Diagrama de Secuencia

Los diagramas de secuencia muestran un flujo detallado para un caso de uso específico o para parte de uno. Generalmente se explican a sí mismos, muestran las llamadas entre los objetos en la secuencia que ocurren.

Estos diagramas poseen dos dimensiones [IBM, 2003]:

- Dimensión Vertical: muestra la secuencia de mensajes o llamadas en el orden temporal que ocurren.
- Dimensión Horizontal: muestra las instancias de los objetos hacia los que son enviados los mensajes.

Un diagrama de secuencia se lee comenzando desde la esquina superior izquierda la instancia de la clase que comience con la secuencia y, a continuación, se sigue cada mensaje hacia abajo.

5.1.1.1.4 Diagrama de Estados

Este diagrama modela los diferentes estados que una clase puede tener y como esta pasa de un estado a otro. Se puede decir que cada clase tiene un estado, pero que no todas ellas deben tener un diagrama de estados asociados. Sólo las clases con estados "interesantes", es decir, clases con 3 o más estados potenciales durante la actividad del sistema, deben ser modeladas [IBM, 2003].

Un diagrama de estados tiene 5 elementos básicos:

- Estado inicial: se representa con un círculo sólido.
- Transiciones entre estados: se dibuja con una flecha.
- Estados: se representa a través de un rectángulo con las esquinas redondeadas.
- Puntos de decisión: se representan con un círculo abierto.
- Estado final: se representa con un círculo sólido dentro de otro círculo abierto.

5.1.1.1.5 Diagrama de Actividades

Los diagramas de actividades muestran el flujo de control entre dos o más objetos de clases, mientras se procesa una actividad. Pueden ser usados para modelar procesos de negocio de alto nivel al nivel de unidades de negocio o para modelar acciones internas de una clase a bajo nivel.

En general estos diagramas son mejor utilizados para modelar procesos de alto nivel, por ejemplo para modelar cómo una compañía hace sus negocios o cómo le gustaría hacerlo. Esto es porque los diagramas de actividad son aparentemente menos técnicos, comparados con los diagramas de secuencia, por lo que personas de las áreas de negocio tienden a entenderlos más rápidamente [IBM, 2003].

5.1.1.1.6 Diagrama de Componentes

Este diagrama entrega una vista física del sistema. El propósito es mostrar las dependencias que el software tiene con otros componentes (por ejemplo, librerías) en el sistema. El diagrama puede ser mostrado a un muy alto nivel, con sólo los elementos más "grandes", o puede ser mostrado a un nivel de componentes de los *packages*.

5.1.2 PERFILES

Una de las ventajas de UML, al ser un lenguaje de propósito general, es su flexibilidad para modelar sistemas pertenecientes a los más diversos dominios, pero esto mismo puede llegar a ser una desventaja cuando se desean modelar sistemas demasiado específicos. En estos casos la sintaxis estándar de UML puede no permitir expresar todos los detalles que el dominio en el cual se está trabajando requiere.

Para definir un lenguaje para un dominio en particular, se tienen dos posibilidades [OMG, 2005]:

- Definir un nuevo lenguaje, alternativo a UML.
- Extender UML, especializando o restringiendo algunos de sus conceptos, respetando la semántica propia del lenguaje.

Cada una de las alternativas presenta ventajas y desventajas, ya que definir un nuevo lenguaje permite un mayor grado expresividad y correspondencia con los conceptos del dominio de aplicación particular. Sin embargo, el hecho de no respetar el metamodelo estándar de UML va a impedir que las herramientas existentes en el mercado para UML puedan manejar estos nuevos conceptos de forma natural. Resulta difícil decidir cuándo se debe crear un nuevo metamodelo y cuando es mejor definir una extensión de UML usando los mecanismos de extensión estándares definidos con este propósito [Fuentes, et al., 2004].

UML 2.0 señala varias razones por las que un diseñador puede querer extender y adaptar un metamodelo existente [Fuentes, et al., 2004]:

- Disponer de una terminología y vocabulario propio de un dominio de aplicación o de una plataforma de implementación concreta.
- Definir una sintaxis para construcciones que no cuentan con una notación propia (como sucede con las acciones).

- Definir una nueva notación para símbolos ya existentes, más acorde con el dominio de la aplicación objetivo (poder usar, por ejemplo, una figura con un computador en lugar del símbolo para representar un nodo que por defecto ofrece UML para representar computadores en una red).

- Añadir cierta semántica que no aparece determinada de forma precisa en el metamodelo (por ejemplo, la incorporación de prioridades en la recepción de señales en una máquina de estados de UML).

- Añadir cierta semántica que no existe en el metamodelo (por ejemplo relojes, tiempo continuo, etc.).

- Añadir restricciones a las existentes en el metamodelo, restringiendo su forma de utilización (por ejemplo, impidiendo que ciertas acciones se ejecuten en paralelo dentro de una transición, o forzando la existencia de ciertas asociaciones entre las clases de un modelo).

- Añadir información que puede ser útil a la hora de transformar el modelo a otros modelos, o a código.

5.2 PERFILES EXISTENTES

Existen algunas propuestas desarrolladas de perfiles UML, las cuales permiten modelar sistemas seguros de una manera más clara y precisa.

5.2.1 CORAS

CORAS es un perfil UML desarrollado para el modelado de amenazas y evaluaciones de riesgos del sistema objetivo. Como se describe en [Soldal, et al., 2003], este perfil utiliza los modelos UML para tres diferentes propósitos:

- **Para describir el objeto a evaluar en un correcto nivel de abstracción:** una evaluación adecuada de la documentación técnica del proyecto no es suficiente, ya que la comprensión del uso del sistema y su rol dentro de la organización o empresa es tanto o más importante. UML permite que estos aspectos sean documentados de una manera uniforme.

- **Para facilitar la comunicación e interacción entre diferentes grupos de actores involucrados en una evaluación de seguridad:** uno de los mayores desafíos cuando se está realizando una evaluación de seguridad es establecer una comprensión común del objeto a evaluar, amenazas, vulnerabilidades y riesgos de seguridad entre los actores que están participando en la evaluación. Esto motiva la generación de un perfil UML que ayude a facilitar la comunicación durante una evaluación de seguridad, haciendo el diagrama UML mas fácil de comprender para los no expertos y al mismo tiempo manteniendo la buena definición de UML.

- **Para documentar los resultados de la evaluación de seguridad y los supuestos de los cuales estos resultados dependen para soportar la reutilización y mantenimiento:** las evaluaciones de seguridad son costosas y consumen mucho tiempo, por lo que no debieran partir desde cero cada vez que se evalúa un nuevo sistema o uno modificado. Documentar las evaluaciones a través de UML permite la reutilización de la documentación de la evaluación, tanto para sistemas que están bajo mantención como para los nuevos, si evaluaciones similares se han realizado previamente.

El perfil UML creado (CORAS) soporta estos objetivos, pero con un énfasis especial en la comunicación y documentación. La documentación es soportada porque el metamodelo del perfil es consistente con la estructura de datos de las evaluaciones de seguridad de CORAS. La comunicación es soportada por la definición de íconos fáciles de comprender asociados con el modelado de elementos del perfil.

5.2.1.1 METAMODELO

El metamodelo de este perfil está subdividido en 6 submodelos, que soportan las diferentes etapas del proceso de administración de riesgos. Una evaluación de seguridad siempre comienza identificando el contexto de la evaluación, el análisis de fortalezas, debilidades, oportunidades y amenazas (SWOT, por sus siglas en inglés) puede ser parte también de esto. Después de que el contexto ha sido establecido, la parte de evaluación del riesgo puede ser dividida en identificación y documentación de incidentes no deseados, riesgos y mejoras [Soldal, et al., 2003].

- **Contexto:** consiste de todos los actores y activos del sistema que estarán bajo evaluación, sobre los cuales estarán basadas las evaluaciones futuras.
    - **Actor:** una persona u organización que tiene intereses en el sistema evaluado.
    - **Política de Seguridad:** una regla o regulación definida por un actor, relacionada con la seguridad del sistema evaluado. Una política de seguridad puede estar relacionada con aspectos de seguridad como: confidencialidad, integridad, disponibilidad, no repudio, rendiciones de cuentas, autenticidad y confiabilidad, y debe proveer directrices sobre la evaluación.
    - **Criterio de Evaluación de Riesgo:** es un criterio que identifica los riesgos y los evalúa con el fin de decidir si este es aceptable o no.
    - **Activo:** una parte o característica del sistema que tiene valor para uno de los actores, por ejemplo: el nivel de calidad de un servicio.

- **Entidad:** una parte física o abstracta, o característica del sistema bajo evaluación que se convierte en un activo cuando un actor le asigna un valor, por ejemplo: un servicio entregado por el sistema.
- **Valor del Activo:** el valor asignado a un activo por un actor.
- **Definición de Valor:** definición de los tipos de valores para varios valores utilizados en una evaluación, como el valor de un activo.

- **SWOT (Sigla en inglés para Fortalezas, Debilidades, Oportunidades y Amenazas):** es como una parte del establecimiento del contexto de una evaluación de seguridad, pero SWOT es utilizado para definir las direcciones generales de la evaluación y sus resultados son usados indirectamente en futuras evaluaciones.
  - **Activo de la Empresa:** activo de la organización desde un punto de vista estratégico.
  - **Fortaleza de la Empresa:** fortaleza estratégica de la organización.
  - **Debilidad de la Empresa:** debilidad estratégica de la organización.
  - **Oportunidad de la Empresa:** oportunidad estratégica de la organización.
  - **Amenaza de la Empresa:** algo que amenaza la posición estratégica de la organización.
- **Incidente No Deseado:** tiene que ver con la exploración de las amenazas y vulnerabilidades del sistema bajo evaluación, y como estas pueden combinarse y llevar a potenciales incidentes que puedan dañar el sistema.
  - **Agente de Amenaza:** causa potencial de un incidente no deseado, el cual puede resultar en algún daño para el sistema u organización y sus activos. Las amenazas pueden ser tanto internas (hackers o virus) como externas (fallas en el sistema o empleados desleales).
  - **Escenario de Amenaza:** una descripción de cómo una amenaza puede llevar a un incidente no deseado.
  - **Vulnerabilidad:** una debilidad con respecto a un activo o a un grupo de ellos que pueden ser explotadas por una o más amenazas.
  - **Incidente No Deseado:** un evento no deseado que puede reducir el valor de un activo.
  - **Iniciador:** un incidente no deseado que puede llevar a otro incidente.

- **Agente Amenazante:** define una serie de especializaciones del concepto de agente amenazante. El propósito de este modelo es para soportar el modelado de una variedad de amenazas.
    - **Amenaza Humana:** una persona que, con o sin intención, potencialmente puede actuar en una forma que lleve a incidentes no deseados.
        - **Atacante:** un agente amenazante inteligente que lleva a cabo un asalto a la seguridad de un sistema.
        - **Intruso:** una entidad que gana o intenta ganar acceso al sistema o a un recurso del sistema, sin tener autorización para hacerlo.
        - **Oyente:** una persona que realiza escuchas pasivas y secretas.
        - **Hombre en el Medio:** una forma de ataque de escucha activo en el cual el atacante intercepta y selectivamente modifica los datos comunicados para enmascararse como una o más de las entidades involucradas en la comunicación.
        - **Atacante Interno:** una entidad dentro del perímetro de seguridad, es decir, una entidad que está autorizada para acceder a los recursos del sistema, pero que lo utiliza de una manera no apropiada para los que le concedieron el permiso.
    - **Amenaza del Sistema:** una parte del sistema bajo evaluación que potencialmente puede actuar de una manera que puede llevar a producir incidentes no deseados.
        - **Falla de Hardware:** error de hardware que puede constituir una amenaza.
        - **Falla de Software:** error de software que puede constituir una amenaza.
    - **Software Malicioso:** software hecho con la intención de dañar sistemas computarizados.
        - **Virus:** una sección de software escondida y que se puede replicar a sí misma, usualmente con una lógica maliciosa que se propaga bajo infección.
        - **Gusano:** un programa computacional que puede ejecutarse de manera independiente, puede propagar una versión operativa completa de si mismo hacia otros equipos en la red y puede consumir recursos del sistema de forma destructiva.

- **Zombie:** un programa que secretamente toma otro equipo conectado a Internet y lo utiliza como un computador para lanzar ataques que son difíciles de rastrear hasta el creador del *zombie*.
- **Troyano:** un programa que aparenta tener una función útil, pero que también tiene otra función escondida y potencialmente peligrosa que evita los mecanismos de seguridad.
- **Bomba Lógica:** lógica maliciosa que se activa cuando se cumplen condiciones específicas. Usualmente se utiliza para generar ataques de denegación de servicio u otra forma de dañar los recursos de un sistema.
- **Puerta Trasera:** una falla escondida conocida por un intruso o un mecanismo escondido instalado por el mismo, el cual puede activar una entrada para ganar acceso al computador sin poder ser bloqueado por los mecanismos de seguridad.

- **Riesgos:** es un incidente no deseado al cual se lee han asignado valores de consecuencia y frecuencia. Estos valores se utilizan para calcular el valor del riesgo, el cual representa la pérdida de valor del activo al cual el riesgo está relacionado.
  - **Riesgo Abstracto:** las propiedades comunes de los riesgos y los temas de riesgo, como el valor del riesgo.
  - **Riesgo:** un incidente no deseado al cual se le han asignado valores de consecuencia, frecuencia y de resultado. Amenazas, vulnerabilidades e incidentes no deseados puede ir hacia muchos activos, pero como un riesgo puede reducir el valor de un activo, un riesgo está relacionado sólo a un activo en particular.
  - **Tema de Riesgo:** una categorización de riesgos similares, al cual se le asigna un solo valor de riesgo.
  - **Relación de Riesgo:** la relación entre riesgos o temas de riesgo.
  - **Evaluación de Riesgo:** la asignación de un riesgo o un tema de riesgo a un incidente no deseado que se evalúa con respecto al valor del riesgo.
  - **Consecuencia:** la consecuencia de la ocurrencia de un incidente no deseado, relacionado con un activo.

- **Frecuencia:** la probabilidad de ocurrencia de un incidente no deseado con respecto a un período de tiempo.
- **Valor de Riesgo:** valor asignado a un riesgo, reflejando la pérdida del valor de un activo que es representado por el riesgo.

- **Corrección:** está relacionado con la documentación y evaluación de las formas de proveer mejoras al sistema que está bajo evaluación, con el fin de reducir el valor de los riesgos.
  - **Corrección:** formas de reducir el valor de un riesgo o un tema de riesgos.
  - **Efecto de la Corrección:** capacidad de la corrección para reducir el valor de un riesgo en particular.
  - **Evaluación de la Corrección:** la asignación del efecto de una corrección a la corrección que está evaluando.
  - **Reducción de Riesgo:** el valor del efecto de una corrección.
  - **Opción de Corrección:** las principales clases de entrega de una corrección, es utilizada para asignar una corrección a un incidente no deseado. Las opciones son: evitar, reducir consecuencia, reducir probabilidad y transferir (un riesgo).

5.2.1.2 PERFIL CORAS

Este perfil provee una extensión al metamodelo de UML, introduciendo elementos de modelado para conceptos definidos en el metamodelo de evaluaciones de seguridad, definiendo estereotipos.

No todas las clases del metamodelo tienen asignados estereotipos, más específicamente: políticas de seguridad, definición de valor, valor del activo, criterio de evaluación de riesgo, consecuencia, frecuencia, valor del riesgo y reducción de riesgo. En otras palabras, se dejó indefinido el modelado de estos conceptos. La razón para hacer esto es que estas clases representar restricciones y valores, y no necesitan una representación gráfica explícita. En todo caso, hay muchas maneras en la cuales estos valores y restricciones pueden ser modelados, por lo que parece razonable dejar esto así para los usuarios del perfil [Soldal, et al., 2003].

5.2.2 UMLSEC

A continuación se describirá el perfil UMLsec, descrito en [Jürjens, 2002]. Este perfil consiste de una serie de diagramas basados en la versión estándar de UML, los cuales describen diferentes vistas de un sistema:

- Diagrama de Casos de uso.
- Diagrama de Actividades.
- Diagrama de Clases.
- Diagrama de Secuencia.
- Diagrama de Estados.
- Diagrama de Despliegue.

Este perfil se basa en los mecanismos de extensión que provee UML, los cuales consisten en etiquetas. Estos pueden ser estereotipos (escrito de la forma <<estereotipo>>) o pares con valores etiquetados (escritos de la forma {etiqueta=valor}). Utilizando perfiles se pueden dar significados especiales a elementos del modelo marcados con estas etiquetas. En el caso de UMLsec se utilizan estas etiquetas para expresar requerimientos de seguridad.

Utiliza una combinación de un proceso dirigido por los casos de uso con un enfoque orientado a los objetivos. Se desarrolló un árbol de objetivos de seguridad junto con el desarrollo de la especificación del sistema. Los objetivos de seguridad son definidos en paralelo, entregando más detalles del sistema a las etapas siguientes del desarrollo, siendo, en general, un proceso iterativo.

Ya que se utilizan fragmentos formales de UML, se puede pensar formalmente, por ejemplo, que un sistema es tan seguro como lo son sus componentes. De esta manera, se puede reducir la seguridad de un sistema a la seguridad de los mecanismos que se utilizan.

El perfil es utilizado de la siguiente manera:

1. **Captura de Requerimientos:** se utilizan casos de uso para la captura de requerimientos, describiendo por medio de un estereotipo la semántica del caso de uso, permitiendo detallar de una mejor manera el contexto en que opera dicho requerimiento. En la Figura 5.1 se puede ver un ejemplo de caso de uso con UMLSec, en donde se utiliza el estereotipo <<intercambio>> para describir intercambio de algún bien. Este estereotipo tiene asociado una serie de restricciones o características, como que el inicio lo da el evento de compra y el fin es la venta.

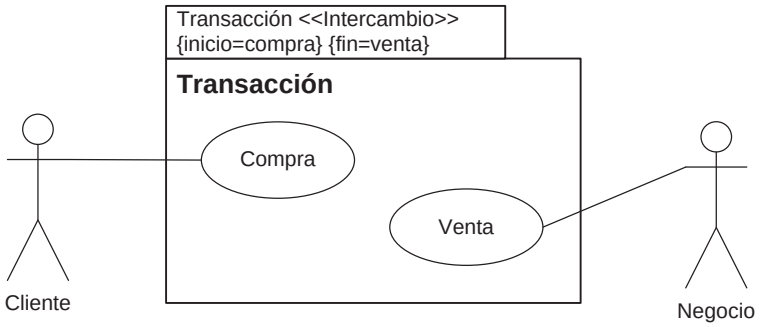

Figura 5.1 Caso de uso de UMLSec.

2. **Análisis:** se utilizan diagramas de actividades para explicar los casos de uso con más detalle. El diagrama es separado en las partes que interactúan con el sistema, además se utilizan pares etiqueta-valor, para marcar ciertas acciones. En la Figura 5.2 se puede ver un ejemplo de diagramas de actividad utilizando UMLSec, en donde se puede ver para el mismo estereotipo <<intercambio>> el inicio se da con el pago, pero el fin es con reclamo (por no entrega del producto entregado) o el retiro del producto.

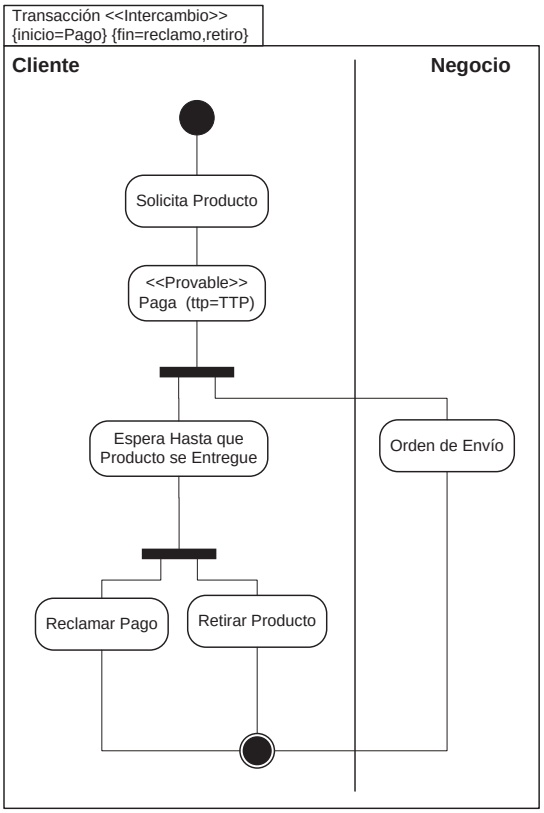

Figura 5.2 Diagrama de actividades UMLSec.

3. **Diseño**

a. **Diagrama de Clases:** se continúan utilizando estereotipos, con el fin de contextualizar cada componente del diagrama, de forma de poder modelar componentes que agreguen seguridad a una aplicación. En la Figura 5.3 se puede ver un ejemplo de diagrama de clases utilizando UMLSec, en donde se ve que se utiliza el estereotipo <<dependencia segura>>, el cual permite reflejar la relación "segura" entre dos clases, ya que en el ejemplo se llama o invoca a un generador de llaves que es utilizado en un generador aleatorio. Esto se utiliza para representar que esa relación debe ser mantenida de forma segura.

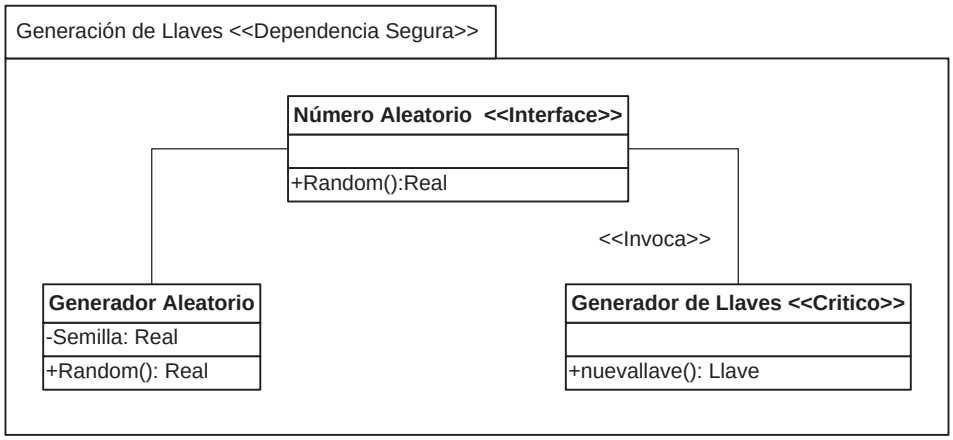

Figura 5.3 Diagrama de clases UMLSec.

b. **Diagrama de Secuencia:** en este caso, es posible especificar protocolos de seguridad utilizando gráficos de secuencia de mensajes, en los cuales se puede detallar de manera se está transmitiendo la información. En la Figura 5.4 se puede ver un ejemplo de diagrama de secuencia utilizando UMLSec, en donde se detalla lo siguiente (asumiendo que se utiliza el algoritmo RSA):

i. Se utiliza el subíndice K para denotar cuando se está encriptando un mensaje.

ii. Se utiliza el subíndice K^{-1} para denotar cuando se está desencriptando o firmando un mensaje.

iii. Se comienza con el mensaje de inicio enviado por el módulo de pago, en donde va la llave se sesión, el cual va firmado por este con su clave privada (K_m^{-1}) y luego encriptado (K_t) con la clave pública de T.

iv. T desencripta el mensaje con su clave privada (K_t^{-1}) y verifica la firma del mensaje recibido con la clave pública de M (K_m).

v. T utiliza la llave de sesión obtenida para encriptar el mensaje que envía hacia el módulo de pago $\{S\}_y$.

vi. El módulo de pago desencripta el mensaje recibió con la llave de sesión que envió a T ($\{S_{SKps}\}$ y responde un OK confirmado la operación.

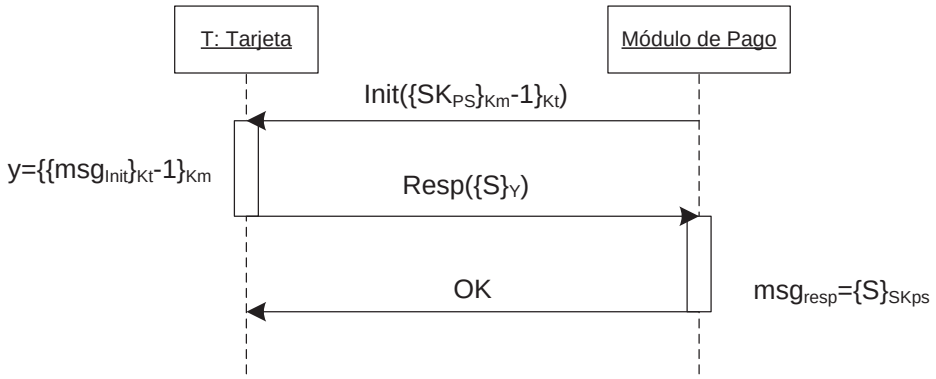

Figura 5.4 Diagrama de secuencia UMLSec.

c. **Diagrama de Estado:** utiliza la misma notación básica de UML para representar los estados de un sistema provocados por el intercambio de mensajes que pueden ser provenientes de protocolos de seguridad, intercambio de llaves, etc. En la Figura 5.5 se puede ver un ejemplo de diagrama de estados con UMLSec, en donde se detalla cómo opera la restricción de acceso a módulos para un determinado sistema:

i. Los elementos entre '[ ]' hacen referencia eventos.

ii. Los elementos que son precedidos por un '\' hacen referencia a una acción.

iii. Los eventos pueden ser una invocación a un método, como por ejemplo: la llamada a checkGuard() en la Figura 5.5.

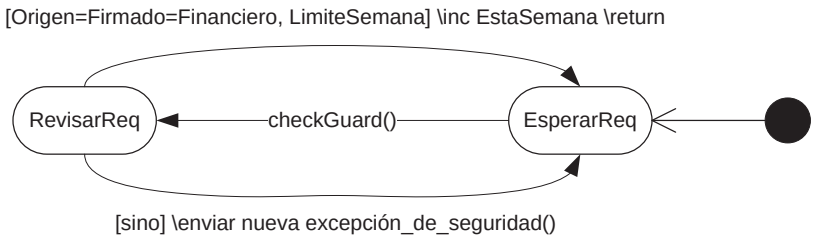

Figura 5.5 Diagrama de estados UMLSec.

d. **Diagrama de Componentes:** estos diagramas pueden ser utilizados para agrupar partes de un sistema en unidades de más alto nivel. Juegan un rol importante en la seguridad de un sistema ya que pueden especificar la visibilidad de los objetos dentro de un paquete desde objetos que estén fuera. En la Figura 5.6 se puede ver un ejemplo de diagrama de paquetes con UMLSec, en donde se detalla cómo interactúan las clases entre dos paquetes:

   i. Cls1 y Cls2a son públicas, por lo que se pueden comunicar.

   ii. Cls2b no puede ser accedida por Cls1, ya que es privada, a pesar de que los dos paquetes estén comunicados.

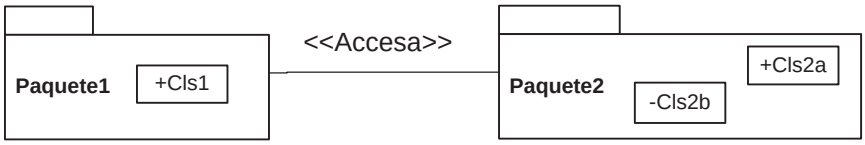

Figura 5.6 Diagrama de componentes UMLSec.

4. **Implementación:** los diagramas de despliegue son utilizados para verificar que los requerimientos de seguridad se cumplen a nivel de la capa física, ya que pueden detallar la comunicación entre componentes independientes de un sistema a nivel de la arquitectura del mismo. Por lo tanto son especialmente útiles cuando estos están físicamente separados. En la Figura 5.7 se puede ver un ejemplo de diagrama de implementación con UMLSec, en donde se detalla como el módulo de control de acceso invoca al método obtener_clave, lo que hace a través de intenet, por lo que es necesario proteger la comunicación, cosa que se hace especifica con la etiqueta <<Secreto>> sobre la línea que representa dicho contacto.

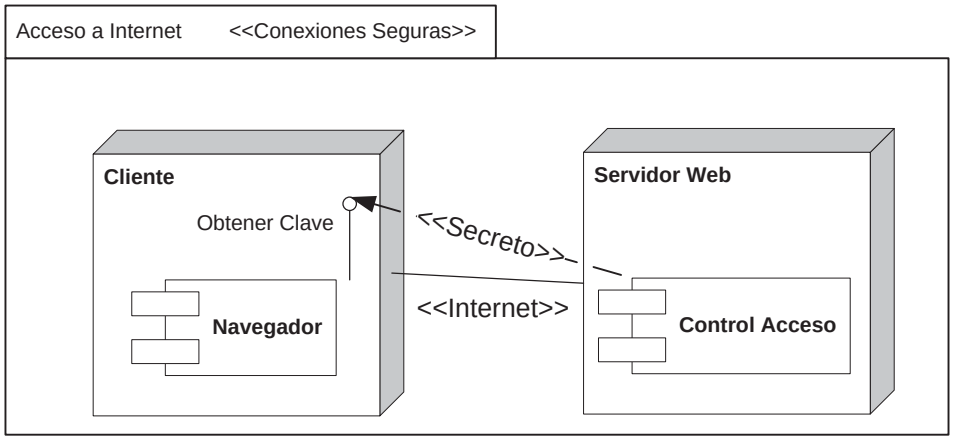

Figura 5.7 Diagrama de implementación UMLSec.

5.2.3 DISCUSIÓN SOBRE LOS PERFILES UML MENCIONADOS

Ambos perfiles UML mencionados anteriormente facilitan en gran medida la incorporación de conceptos de seguridad al lenguaje UML, pero cada uno tiene algunas desventajas que dificultan su aplicación.

- **CORAS**
    - Es un perfil orientado al modelado de amenazas y riesgos, es decir, no está pensado para el análisis y diseño de sistemas seguros.
    - Es más utilizado para las evaluaciones de seguridad y no para el desarrollo.
    - Su objetivo es la documentación de las evaluaciones de seguridad, pero no aportar al proceso de desarrollo de un sistema. Si bien lo generado como resultado del proceso puede servir para un nuevo desarrollo futuro, esta puede ser sólo como apoyo, no como parte de este.
- **UMLsec:** este un perfil muy completo, pero por esa misma razón algunas veces incluye demasiada información o la muestra de una manera en que no es fácil seguirla y necesita lectura adicional para comprender lo que se quiere decir. A continuación se muestran algunos comentarios referentes al uso de este perfil.
    - Diagrama de Casos de Uso
        - No quedan totalmente claros los requerimientos de seguridad, ni qué partes de un sistema se necesitan que cumplan ciertos requisitos en este sentido.

- No se deja de manera explícita cuándo el actor que está interactuando con el sistema es alguien conocido o si el sistema prestará servicios a cualquier usuario que acceda a éste.

- Diagrama de Secuencia

  - La notación no ayuda a una fácil lectura, ya que las llaves con subíndices no son fáciles de leer cuando se agrupan muchas operaciones en un mismo mensaje.

  - No considera incluir en los mensajes que se envían entre las clases la definición del algoritmo de encriptación, debiendo definir como supuesto o asumiendo cual es el que se está utilizando. El problema es que en etapa de diseño, ya es posible hacer este tipo de definiciones, pero el modelo no es capaz de describirlo.

- Diagrama de Clases:

  - No establece definiciones respecto de cómo almacenarán o manejarán los datos mas sensibles los métodos de las clases, por ejemplo: cómo se manejarán las claves de los usuarios.

  - No se entrega información sobre el volumen de datos que manejará cada clase. Este dato es importante, ya que no es lo mismo programar un método que manejará un volumen de 10 registros a uno que lo hará sobre 1.000.000 de registros.

- Diagrama de Estados

  - Si bien, hace referencia a las transiciones, ya sea por eventos o acciones, no refleja los permisos para esos cambios, ya que es importante para las transiciones de estado en un sistema conocer si existen restricciones en este sentido.

- Diagrama de Componentes

  - Este diagrama no diferencia entre componentes que interactúan con el usuario y los que no. Este es un punto importante, ya que si un módulo recibe datos directamente desde el navegador, se deben realizar validaciones muy diferentes sobre las entradas de datos, a diferencia de los módulos internos sin comunicación con el "exterior".

- Algunas veces los componentes no están en un mismo equipo físico, por lo que es importante la forma en que se comunicarán y qué datos se enviarán. Este diagrama no considera definir restricciones de diseño en cuanto a métodos de seguridad para el envío de información, ni tampoco qué es lo que se envía. Esto último también es importante, ya que el nivel de seguridad que se implementa depende de los datos que se transmiten.

6 PERFIL UML PARA DESARROLLO DE SISTEMAS SEGUROS

6.1 PROPUESTA DE PERFIL UML

Con las limitaciones de notación y de modelado de los perfiles UML mencionados anteriormente, se propone un nuevo perfil que mejore dichas limitantes, con el fin de lograr un análisis y diseño que represente de mejor manera los requerimientos y las propuestas de seguridad que permitirán desarrollar una aplicación menos vulnerable.

A continuación se presenta una primera aproximación a la propuesta de perfil UML para desarrollo de sistemas seguros.

6.1.1 DIAGRAMA DE CASOS DE USO

La propuesta de perfil no modifica la estructura de los diagramas, sólo agrega información relevante que es necesario discutir con detalle al momento del diseño de la solución, como lo es la forma de autenticación de los usuarios, los algoritmos criptográficos a utilizar o la cantidad de datos que se espera el sistema administre.

El siguiente es un ejemplo general de utilización de los cambios propuestos y posteriormente se detallan sus componentes nuevos.

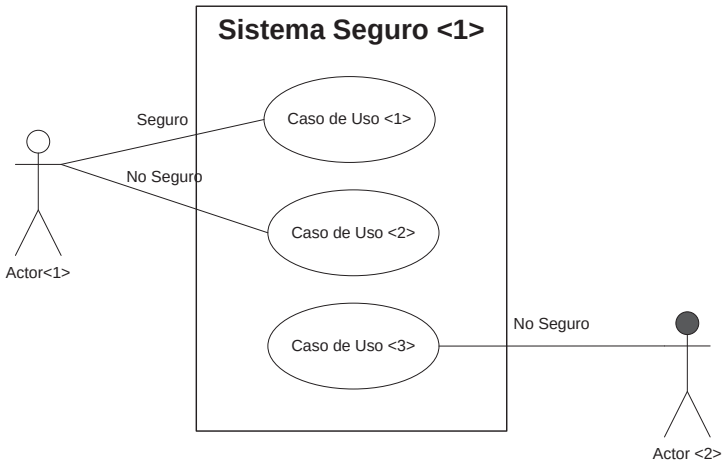

Figura 6.1 Ejemplo de propuesta de diagrama de caso de uso seguro.

Donde cada nuevo componente significa:

- **Actor Conocido:** representa a un usuario autenticado por el sistema, se refiere a usuarios que necesitan realizar un ingreso de usuario y contraseña para operar.

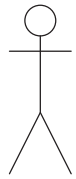

Figura 6.2 Figura que representa a un usuario controlado e identificado.

- **Actor Desconocido:** representa a un usuario desconocido por el sistema, ya que no necesita realizar una identificación previa para interactuar con el sistema.

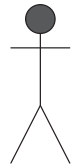

Figura 6.3 Figura que representa a un usuario desconocido por el sistema.

- **Comunicación Segura:** identifica las comunicaciones que deben ser seguras o encriptadas entre el usuario y el sistema.

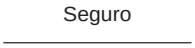

Figura 6.4 Figura que representa una comunicación segura.

- **Comunicación Insegura:** identifica las comunicaciones que pueden realizarse de manera insegura, sin comprometer la integridad del sistema.

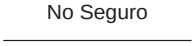

Figura 6.5 Figura que representa una comunicación insegura.

6.1.2 DIAGRAMA DE SECUENCIA

Para el diagrama de secuencia se agregan algunas etiquetas y formas de describir elementos de diseño que permiten especificar con mayor detalle las decisiones de seguridad que se tomen.

El siguiente es un ejemplo general de utilización de los cambios propuestos y posteriormente se detallan sus componentes nuevos.

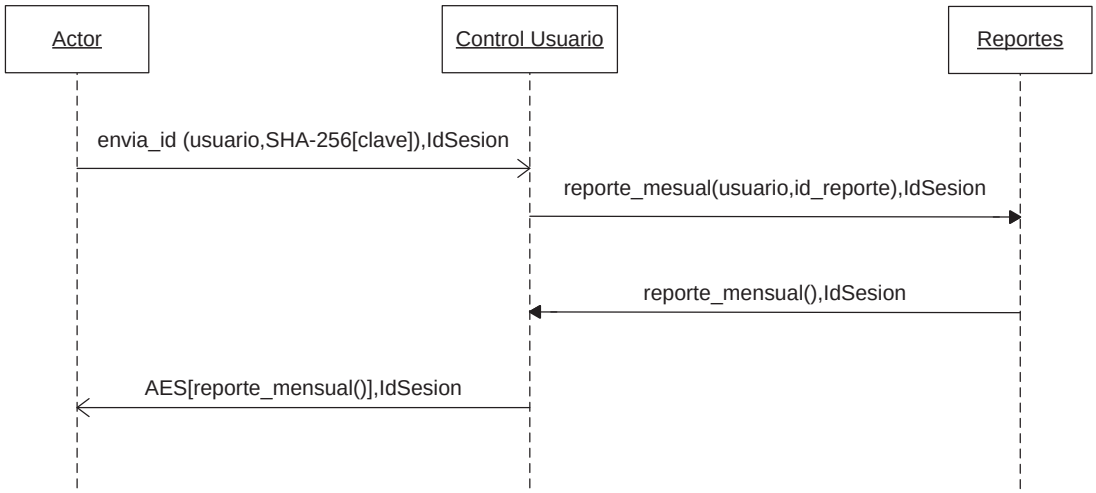

Figura 6.6 Ejemplo de propuesta de diagrama de secuencia seguro.

En el diagrama anterior se utilizaron los siguientes nuevos descriptores:

- **SHA-256[clave]:** esto quiere decir que la clave ha sido procesada por un algoritmo de hash antes de ser enviada al servidor, en donde el algoritmo que se utiliza depende exclusivamente de una decisión de diseño, escogiendo de los que estén disponibles al momento de la definición. Descrito de una forma más general:

FUNCION_HASH[Mensaje]

- **AES[reporte_mensual]:** esto significa que el reporte ha sido encriptado para ser enviado al usuario, el cual debe conocer la clave para poder acceder al contenido. Esto puede ser realizado por diversos algoritmos, no necesariamente con AES (es sólo un ejemplo), pero la etapa de diseño es un buen momento para evaluar cual se utilizará. Otra posibilidad es encriptar toda la conexión por medio de TSL/SSL, lo cual se puede escribir como SSL(mensaje). Descrito de una forma más general:

CIFRADO]Mensaje]

- **TOKEN:** se refiere a un identificador único de sesión, similar a un SESSIONID. Es utilizado para mantener identificada la sesión y por consiguiente, al usuario que está interactuando con la aplicación. Este puede ser numérico, alfabético o alfanumérico y es de largo arbitrario. Es particularmente útil cuando interactúa más de una aplicación, ya que entre ellas se pierde el tradicional id de sesión que se utiliza de forma estándar.

6.1.3 DIAGRAMA DE CLASES

Para este diagrama se propone describir los campos que se desean o necesita almacenar encriptados, de una forma en que quede claro como se desea hacer.

En la Figura 6.7, se muestra como se detalla el uso de una función hash para el almacenamiento de las claves de usuario, donde SHA-256[Clave] quiere decir que se utiliza el algoritmo SHA-256 para procesar el campo Clave.

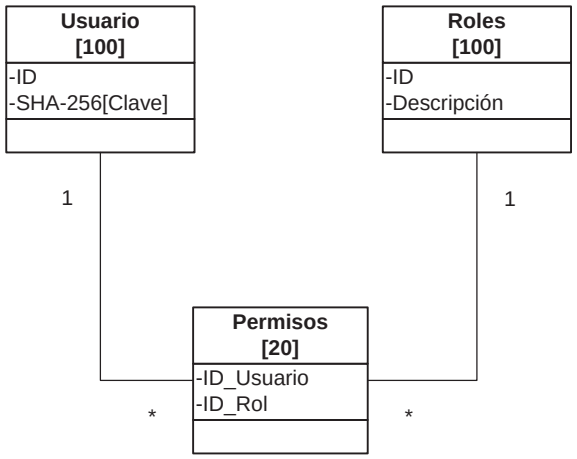

Figura 6.7 Ejemplo de propuesta de diagrama de clases seguro.

De una forma más general esto se denota como:

ALGORITMO_CIFRADO_O_HASH[Clave]

Además, junto con el nombre de las clases, como se puede ver en la misma Figura 6.7, se agrega información respecto de la cantidad de datos esperados que cada clase deberá manejar, lo cual en el caso del ejemplo sería:

- Usuario: 100 datos
- Roles: 100 datos
- Permisos: 20 datos

Esto es importante, ya que la forma en que se implementarán los métodos de cada una de las clases puede variar considerablemente dependiendo de la cantidad de datos que debe procesar sin afectar los tiempos de respuesta o generar errores, lo cual puede provocar problemas de seguridad y/o estabilidad. Una mala implementación en este sentido puede provocar que una aplicación sea más sensible a un ataque de denegación de servicio.

Obviamente hay casos en los cuales no es necesario agregar este dato (como en la clase Permisos), porque la cantidad de datos que manejará es baja. Por esto último no es obligatorio agregar esta información todas las clases, pero sí lo es para las que se estime deberán manejar grandes volúmenes de datos.

6.1.4 DIAGRAMA DE ESTADOS

El objetivo de este diagrama es reflejar los diversos estados por los que pasa un sistema, pero el problema es que no es capaz de definir quién (persona, rol o cargo) puede hacer que el sistema, módulo o entidad cambie de estado. Esto se puede mejorar agregando un sistema de etiquetado, con el cual se define quien puede y no puede hacer que algún sistema, módulo o entidad pase de un determinado estado a otro.

En la Figura 6.8, se muestra un ejemplo de utilización de este etiquetado. Este se refiere a algunos estados que puede tomar un proyecto en el proceso de desarrollo desde el estado inicial hasta el final. Cada cambio de estado esta etiquetado con el (en este caso) rol que puede realizarlo u ordenarlo.

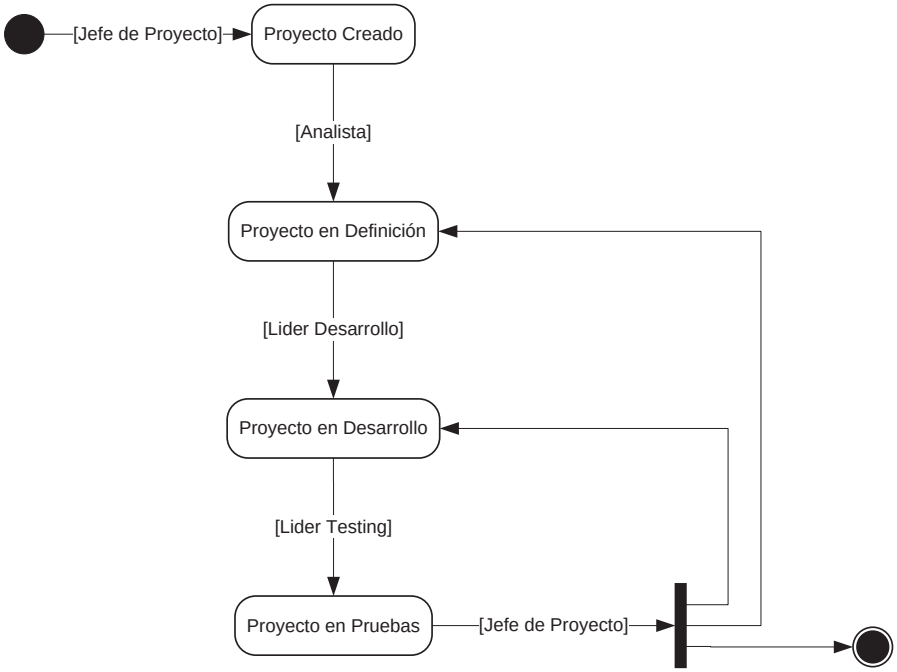

Figura 6.8 Ejemplo de propuesta de diagrama de estados seguro.

Además de reflejar con las etiquetas quién puede realizar los cambios de estado, también es posible representar quien no puede hacerlo, si lo que se desea hacer es expresar las restricciones:

- [autorizado]: esta notación se utiliza para cuando se quiere representar a alguien autorizado a realizar el cambio de estado.

- [no autorizado]: esta notación se utiliza para cuando se quiere representar a alguien que no puede realizar el cambio de estado.

6.1.5 DIAGRAMA DE COMPONENTES

Muchas veces no sólo basta con detallar los módulos que intervienen en un sistema, sino que también cómo estos se comunican entre sí y cuáles son los que interactúan con el usuario, es decir, tienen una interfaz directa con éste.

Para esto se agregaron etiquetas al diagrama estándar que detallan la comunicación entre los módulos, junto con una forma de destacar los que tienen una interfaz hacia el usuario final.

En la Figura 6.9 se muestra un ejemplo para un sistema relativamente estándar, con 5 módulos y sus interacciones.

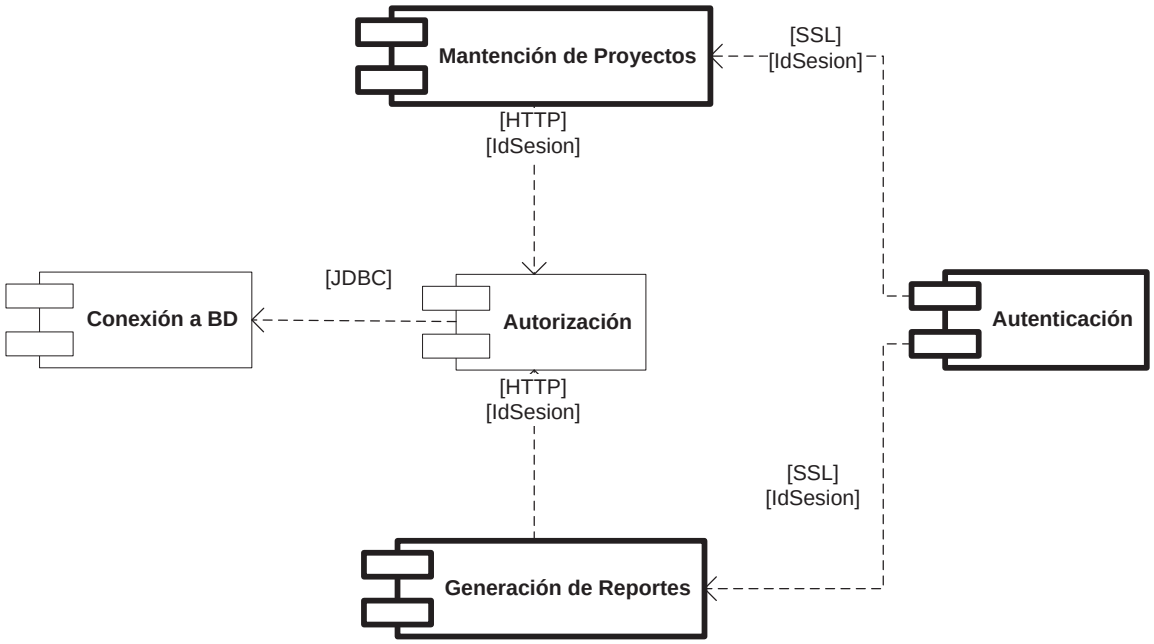

Figura 6.9 Ejemplo de propuesta de diagrama de componentes seguro.

En el ejemplo el módulo de autenticación se comunica con los módulos de mantención de proyectos y generación de reportes a través de SSL y estos dos últimos se comunican a su vez con el módulo de autorización sólo vía HTTP para luego acceder a la base de datos. Todos los módulos aseguran que están utilizando una sesión válida a través de un *IdSesion* que se adjunta en cada comunicación. Además los módulos de autenticación, mantención de proyectos y generación de reportes tienen interfaces con el usuario, lo cual queda representado a través de una línea más gruesa en el diagrama. Esto último, con el fin de graficar el mayor cuidado que hay que tener al codificarlos, ya

que (como se ha discutido anteriormente) es muy importante controlar todas las entradas de usuario, cosa que no ocurre con los módulos de autorización y conexión a la base de datos.

Finalmente lo que se propone para este diagrama es lo siguiente:

- Identificar, mediante una línea más gruesa para el cuadro que representa al módulo, cada uno de los cuales que recibe datos directamente desde el usuario.

- En cada línea que representa la comunicación entre módulos, agregar etiquetas que reflejen como se realizará esta comunicación. Hacer esto de la forma: [COMUNICACION], en donde se debe definir si ésta se realizará cifrada y de qué forma o si se realizara sin cifrado de datos.

7 CASO PRÁCTICO DE ESTUDIO

Con el fin de demostrar la utilidad de lo propuesto en el presente trabajo, se utilizará lo expuesto, poniéndolo en práctica con un problema real. A continuación se describirá dicho problema, detallando sus características generales, requerimientos y alcance.

7.1 PROBLEMA

En el desarrollo de un sistema, durante la ejecución de las pruebas funcionales es necesario tener un control y seguimiento de los casos de prueba ejecutados correctamente o con error, además del seguimiento de los incidentes generados, concepto utilizado para representar los errores encontrados al momento de las pruebas de un nuevo desarrollo. En el proceso de corrección de incidentes interviene el área de desarrollo, el cual debe recibir dichos incidentes, con la información necesaria para reproducirlos y emitir una respuesta, la cual puede ser una corrección, un no aplica (no existe el error), no se puede reproducir el error, etc.

Los jefes de proyecto encargados del desarrollo y de las pruebas necesitan conocer los estados de avance, la tasa de incidentes encontrados, los que han sido corregidos, los que han sido reportados erróneamente y son rechazados por desarrollo, etc. Para esto requieren reportes de alto nivel que sólo deben ser vistos por ellos, los cuales también pueden incluir tasas de rendimiento de personas individuales.

Durante la ejecución de las pruebas hay que tener especial cuidado con la posibilidad de que se puedan cambiar ciertos datos durante su corrección o la ejecución de las pruebas, ya que en momentos cercanos a las fechas comprometidas es usual que surjan roces entre ambas partes y la presión puede llevar a modificar con fines poco éticos datos importantes del proyecto.

Uno de los problemas centrales en el aspecto de la seguridad es que al estar, generalmente, los equipos en distintas ubicaciones físicas, la aplicación tendrá que estar publicada en Internet, con el fin de garantizar el acceso a todos los involucrados. Esto provocará que se tengan que considerar seriamente las medidas de seguridad y control de acceso, ya que el sistema puede contener información sensible para un cliente y sus sistemas, dependiendo del dominio en el cual trabaje.

En la Figura 7.1 se puede ver un esquema que representa quienes participan y como se realiza la interacción con el sistema de gestión. En éste se puede ver que cada una de las entidades realiza distintas tareas y necesita diferente información del sistema, por lo cual se hace necesario un correcto manejo de los perfiles de usuario. Existen roles que interactúan en ambos sentidos con el sistema, ya que modifican y obtienen datos de él, pero existen otros roles, como los jefes de proyecto y el cliente, que sólo necesitan obtener información como informes de estado y avance.

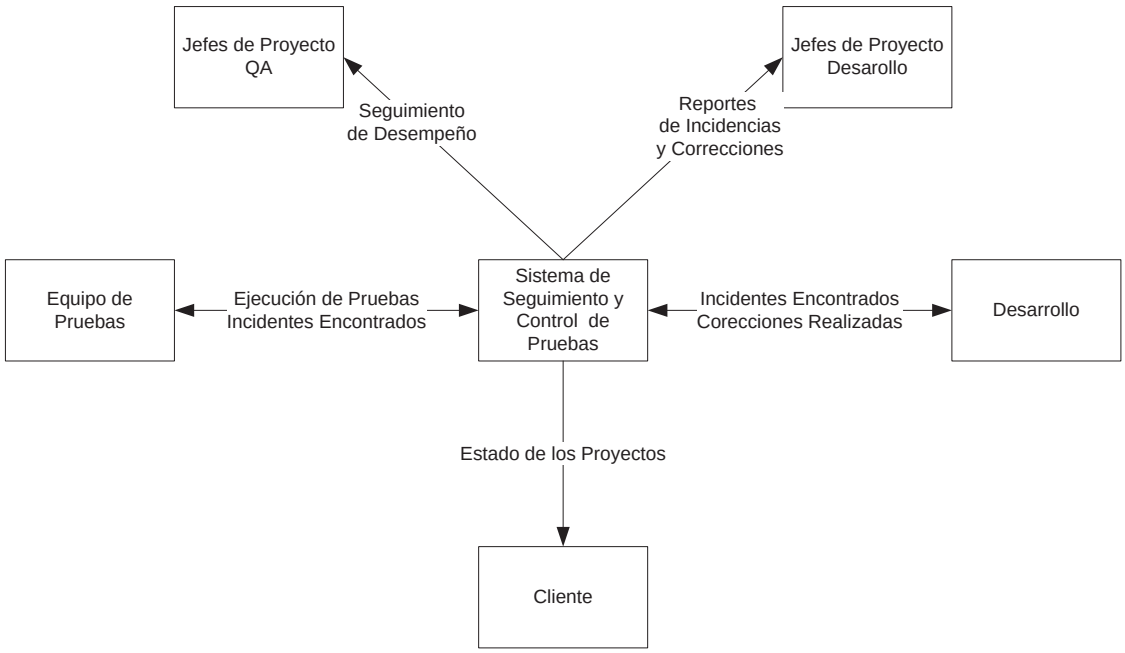

Figura 7.1 Esquema de interacción de sistema de gestión de pruebas.

7.2 DOMINIO Y ALCANCE

El sistema será utilizado por jefes de proyecto de desarrollo y QA, por el equipo de desarrollo encargado de corregir los defectos encontrados y por los testers encargados de ejecutar las pruebas funcionales. Además, es posible dar accesos a jefes de área o gerentes que deseen conocer los estados de avance y situación actual de proyectos críticos.

El sistema sólo se encargará de registrar los eventos y cambios que vayan ocurriendo durante las pruebas, no será capaz de ejecutar pruebas por si solo de forma automatizada.

7.3 REQUERIMIENTOS

A continuación se detalla el listado de requerimientos funcionales y no funcionales del sistema antes descrito. El listado está en la forma en que un usuario lo hubiera entregado:

- Almacenar y administrar el listado de casos de prueba de los sistemas sometidos a pruebas funcionales.
- Permitir llevar un control y seguimiento de los diferentes estados de estos casos de prueba. Los cuales son:
    - **No ejecutado:** el caso está diseñado, pero aún no se ha ejecutado.

- **Ejecutado OK:** el caso de prueba se ejecutó correctamente.
- **Ejecutado con Error:** el caso de prueba se ejecutó con error.
- **No Aplica:** el caso de prueba no corresponde con la funcionalidad.

- Debe llevar un control y seguimiento de los incidentes generados y permitir asociarlos a los casos de prueba que los produjeron. Además controlar sus estados posibles:
  - **Hallado:** estado inicial del incidente, con este estado se notifica a desarrollo que se encontró un error.
  - **Corregido:** el incidente ha sido corregido.
  - **No Aplica:** el incidente no corresponde o fue reportado erróneamente.
- Administrar los distintos perfiles de usuario necesarios al momento de ejecutar las pruebas. Los cuales son:
  - **Cliente:** persona a cargo de los proyectos, en general le interesa ver el estado actual del proyecto y las tasas de avance.
  - **Jefe de Proyecto de Desarrollo:** persona a cargo del proceso de desarrollo de una nueva aplicación o sistema. Está interesado en ver el estado actual del proceso de pruebas y las cantidad de errores que se han detectado y sus respectivas correcciones.
  - **Jefe de Proyecto QA:** persona a cargo de los procesos de certificación. Es la contraparte del jefe de proyecto de desarrollo, por lo que también está interesado en ver el estado actual y las tasas de avance.
  - **Coordinador Pruebas:** persona a cargo del equipo de pruebas, debe poder ver los reportes de avance y el detalle de cada proyecto
  - **Tester:** persona a cargo de ejecutar las pruebas, debe poder cambiar los estados de los casos de prueba y generar incidentes.
  - **Desarrollador:** persona a cargo de programar y corregir los errores, debe poder ver los incidentes encontrados y generar respuestas para ellos.
- Manejar un *workflow* que administre las tareas y cambios que puede realizar cada perfil de usuario.

- Generar reportes de estados de avance, rendimiento, tasas de incidentes encontrados, corregidos, etc.
- Controlar el acceso al sistema a través de nombres de usuario asignados a cada persona que deba acceder a este.
- Cada perfil debe poder configurarse para que tenga accesos a cierta información en particular y controlar que puede y que no puede modificar.
- El sistema estará publicado en internet para que los involucrados en los diversos proyectos tengan acceso a la información.

7.4 DEMOSTRACIÓN DE UTILIZACIÓN DE LO PROPUESTO

A continuación se presenta una demostración de la propuesta de perfil UML presentada en este documento. Se muestran los diagramas más comunes de este lenguaje, junto con una explicación.

7.4.1 DIAGRAMAS DE CASOS DE USO

El primer diagrama de caso de uso, que se puede ver la Figura 7.2, es el que representa al sistema completo. En él se puede ver que diversos actores se relacionan con el sistema, debiendo hacerlo de manera segura, como detallan las líneas que los comunican con los diversos módulos.

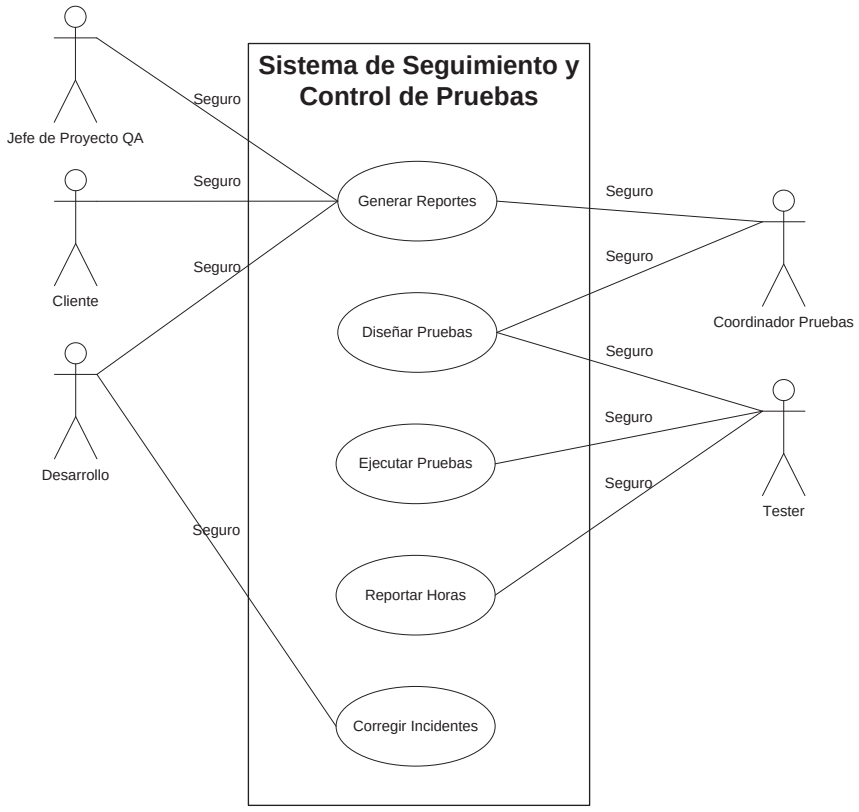

Figura 7.2 Diagrama de caso de uso de sistema.

En la Figura 7.3 se puede ver el caso de uso que representa el ingreso a la aplicación. En él se observa que el proceso de autenticación debe hacerse para todos los usuarios y mediante un canal seguro.

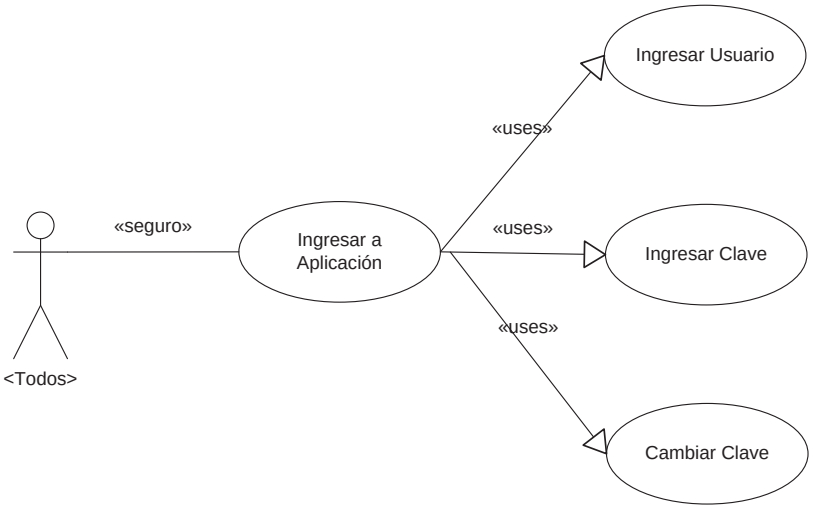

Figura 7.3 Diagrama de caso de uso de ingreso a la aplicación.

En la Figura 7.4 se pueden ver los casos de uso para la generación de reportes, en donde se puede ver que para cada perfil de usuarios, existen reportes distintos que pueden ver, los cuales varían en que información y con qué nivel de detalle se muestran. Además se puede ver que la comunicación que el usuario realiza con el caso de uso de generación de reportes también debe ser segura.

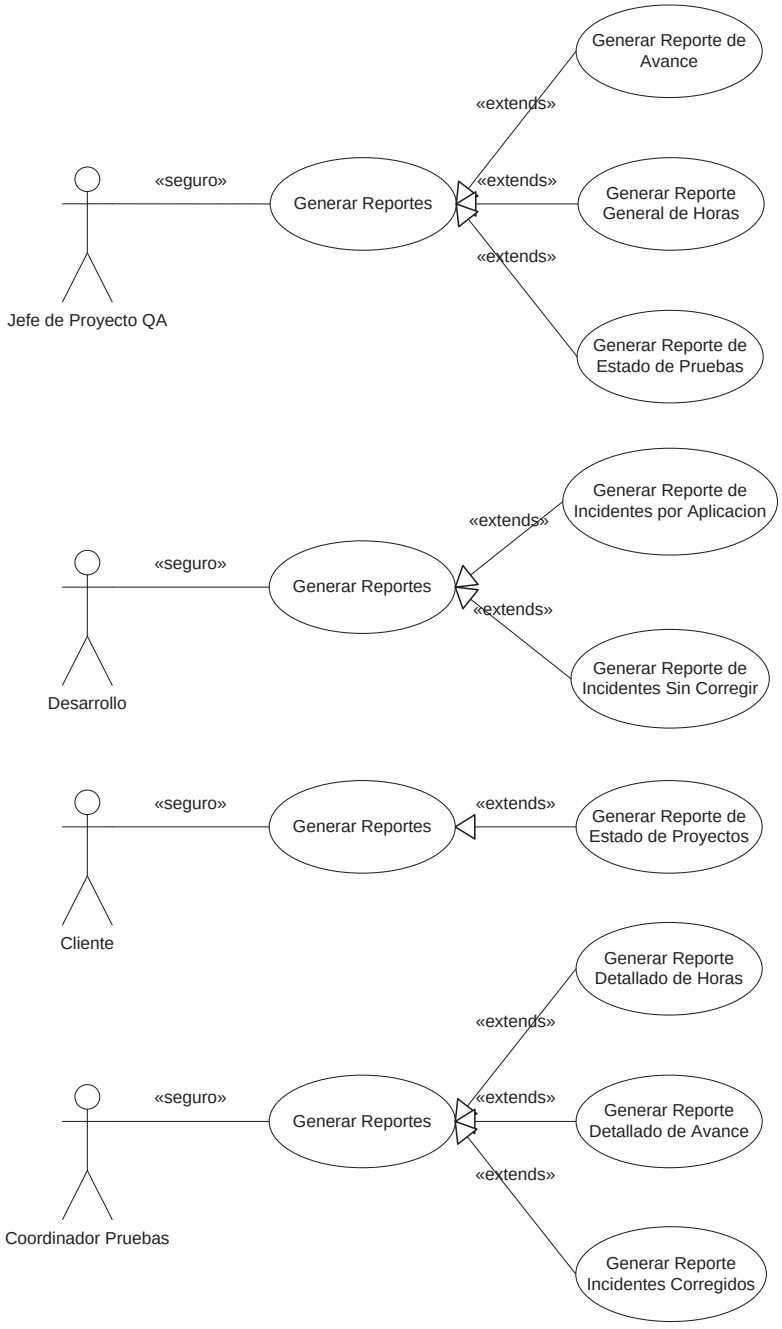

Figura 7.4 Diagramas de caso de uso para la generación de reportes.

En la Figura 7.5 se puede ver el caso de uso para el diseño de pruebas, el cual es realizado por el perfil de "Tester". Además, se ve que el proceso de diseño de casos de prueba también debe ser realizado por un canal seguro, del que se desprenden una serie de casos de uso que dan una idea de lo que se realiza durante el proceso de diseño de las pruebas.

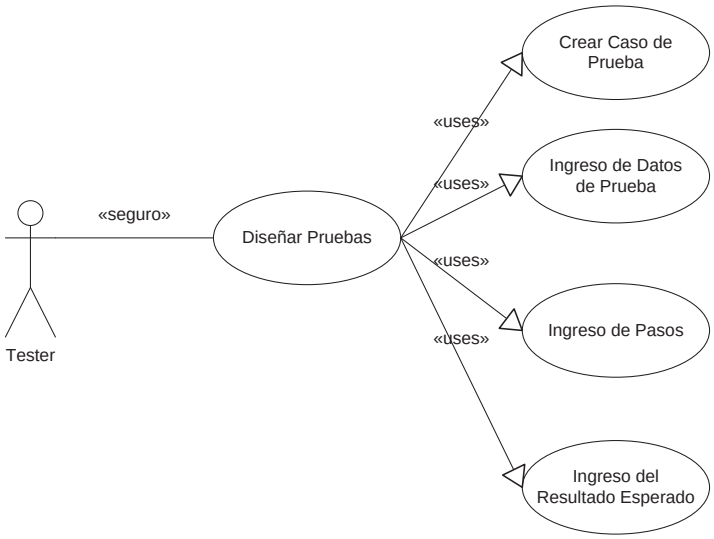

Figura 7.5 Diagrama de caso de uso para el diseño de pruebas del tester.

En la Figura 7.6 se puede ver la interacción del "Coordinador de Pruebas" con el proceso de diseño de estas. Este perfil realiza tareas diferentes a las del "Tester" sobre el caso de uso, las cuales quedan definidas en dicha figura. Además de que se establece que la comunicación es a través de un canal seguro.

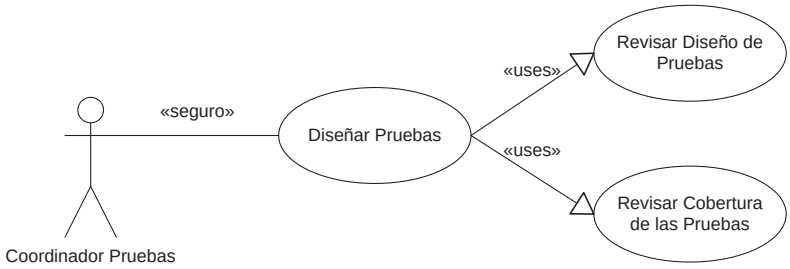

Figura 7.6 Diagrama de caso de uso para el diseño de pruebas del coordinador de las pruebas.

En la Figura 7.7 se puede ver el caso de uso para la ejecución de las pruebas, tarea que realiza el perfil de "Tester", el cual involucra cambios de estado en los casos, como se detalla en la figura. Tanto la comunicación con el caso de uso de ingreso, como con el de ejecución, deben realizar de manera segura.

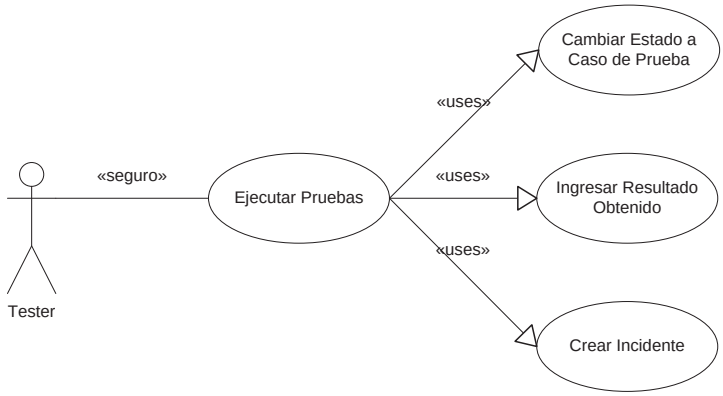

Figura 7.7 Diagrama de caso de uso para la ejecución de las pruebas.

En la Figura 7.8 se puede ver el caso de uso para la creación de incidentes. Este diagrama está relacionado con el de la Figura 7.7, ya que es una especificación del caso de uso "Crear Incidente", por lo que la interacción del usuario comienza desde ese instante. La comunicación segura llega hasta el caso de uso de mantención de incidentes, como se puede ver en la figura.

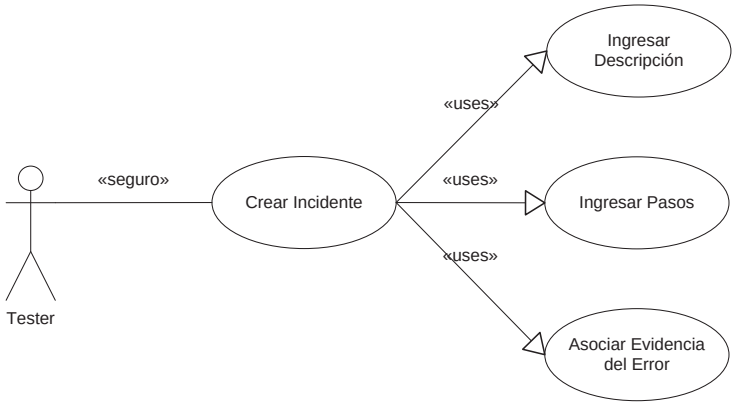

Figura 7.8 Diagrama de caso de uso para la creación de incidentes.

En la Figura 7.9 se puede ver el caso de uso para la corrección de incidentes, tarea que debe realizar el perfil "Desarrollo", como se puede ver la comunicación segura se debe realizar hasta llegar a la corrección de incidentes.

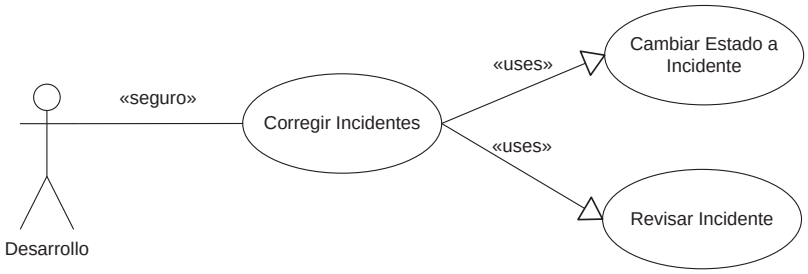

Figura 7.9 Diagrama de caso de uso para la corrección de incidentes.

En la Figura 7.10 se puede ver el caso de uso para el reporte de horas, tarea que debe realizar el perfil "Tester". En la figura se puede ver que el reporte de horas también se debe realizar por un canal seguro.

Figura 7.10 Diagrama de caso de uso para el reporte de horas.

7.4.2 DIAGRAMAS DE SECUENCIA

A continuación se presentan una serie de diagramas de secuencia que cubren los requerimientos del problema planteado.

En la Figura 7.11 se puede ver el diagrama de secuencia para el ingreso a la aplicación. Notar que la conexión segura se utiliza para la interacción con el browser (usuario), sin embargo para la comunicación hacia la base de datos no se utiliza, ya que no es necesaria al no estar expuesta a la red pública. La comunicación segura se realiza mediante el uso de SSL.

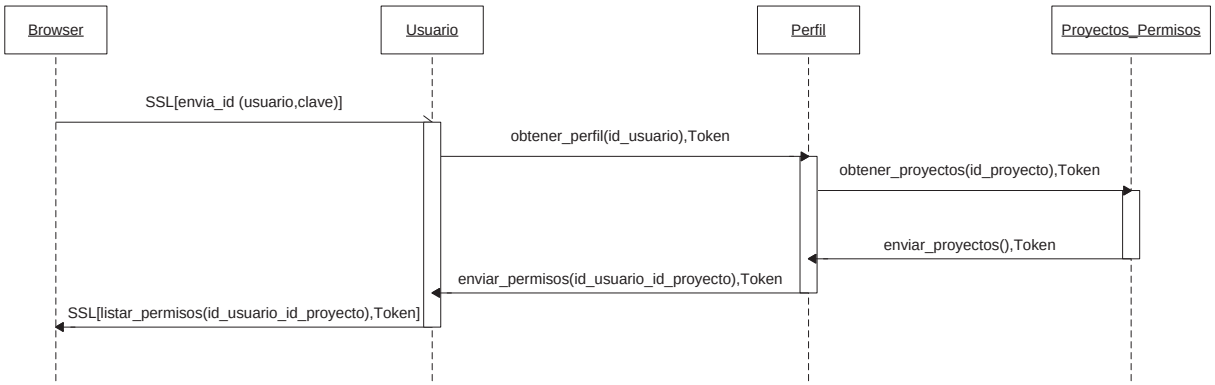

Figura 7.11 Diagrama de secuencia para el ingreso a la aplicación.

En la Figura 7.12 se puede ver el diagrama de secuencia para la generación de reportes, en el se puede ver que todas las comunicaciones que se realizan son mediante un canal seguro, ya que se interactúa directamente con el browser del usuario, primero en la solicitud de permisos y después en la generación del reporte mismo.

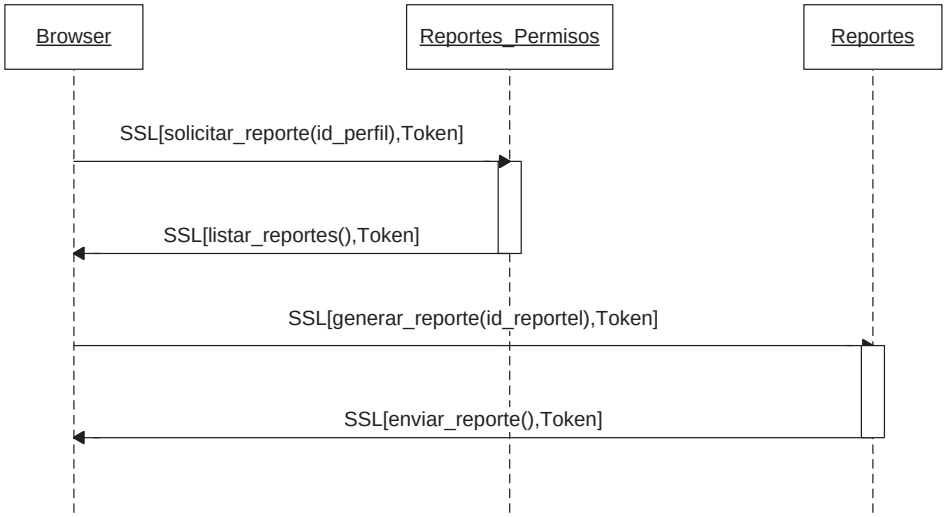

Figura 7.12 Diagrama de secuencia para la generación de reportes.

En la Figura 7.13 se puede ver el diagrama de secuencia para el diseño de las pruebas, donde se puede ver que toda la comunicación debe ser utilizando SSL, ya la interacción es directamente con el browser del usuario.

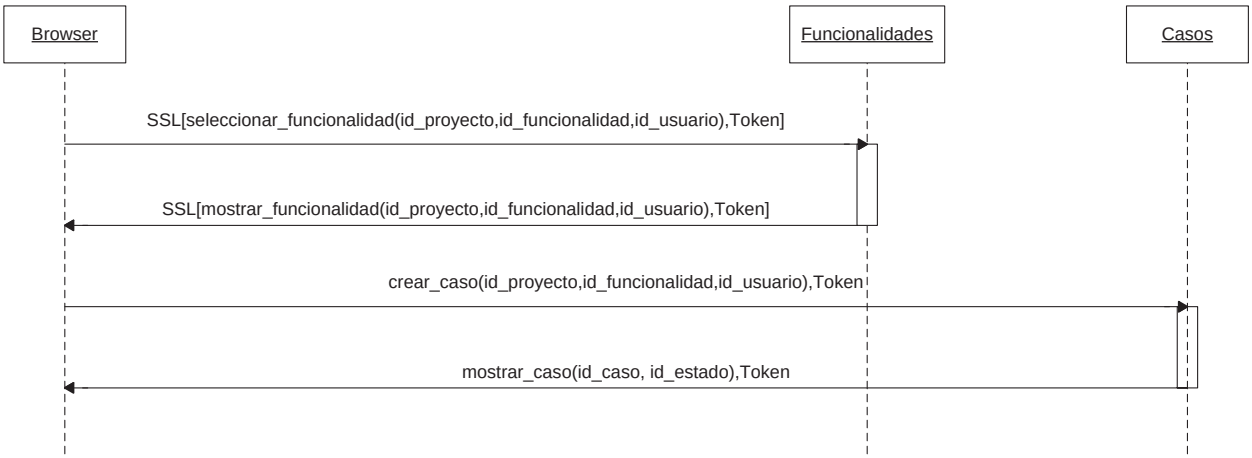

Figura 7.13 Diagrama de secuencia para el diseño de las pruebas.

En la Figura 7.14 se puede ver el diagrama de secuencia para la ejecución de las pruebas, en el se puede ver que toda la comunicación es mediante SSL, menos cuando se consulta el *workflow* para saber si el cambio de estado está autorizado, ya que es una consulta interna, sin que intervenga el usuario.

Figura 7.14 Diagrama de secuencia para la ejecución de las pruebas.

En la Figura 7.15 se puede ver el diagrama de secuencia para la creación de incidentes, en el cual se puede ver que toda la comunicación es mediante un canal seguro, excepto cuando consulta los estados posibles, ya que es una consulta interna de la aplicación.

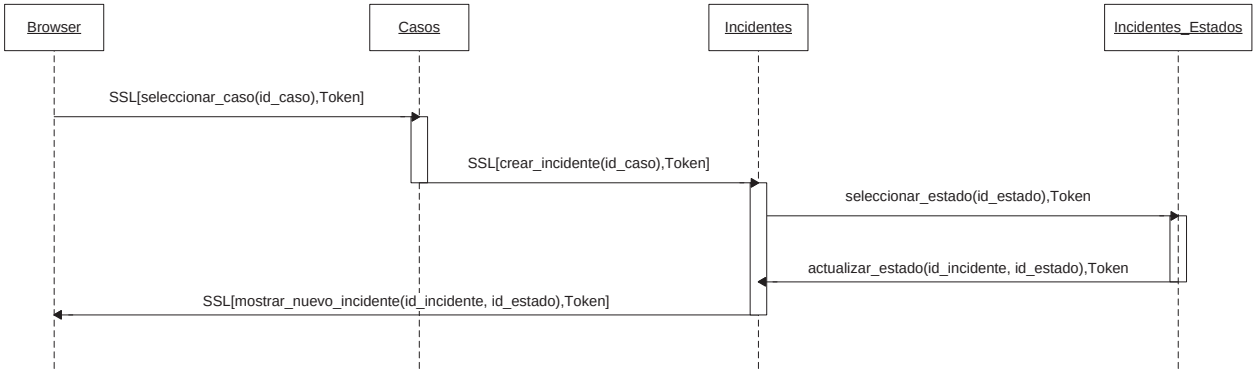

Figura 7.15 Diagrama de secuencia para la creación de incidentes.

En la Figura 7.16 se puede ver el diagrama de secuencia para la corrección de incidentes, en el se puede ver que toda la comunicación es mediante SSL, excepto cuando se consulta el *workflow* para conocer los estados posibles, ya que es una consulta interna de la aplicación.

Figura 7.16 Diagrama de secuencia para la corrección de incidentes.

En la Figura 7.17 se puede ver el diagrama de secuencia para el reporte de horas que debe realizar el perfil de "Tester". En él se puede ver que la primera parte de la interacción es mediante SSL, ya que el usuario es el que ingresa las horas y la tarea, pero después internamente se identifica dicha tarea y se ingresa el registro, por lo que esa parte no es necesario que sea realizada por un canal seguro.

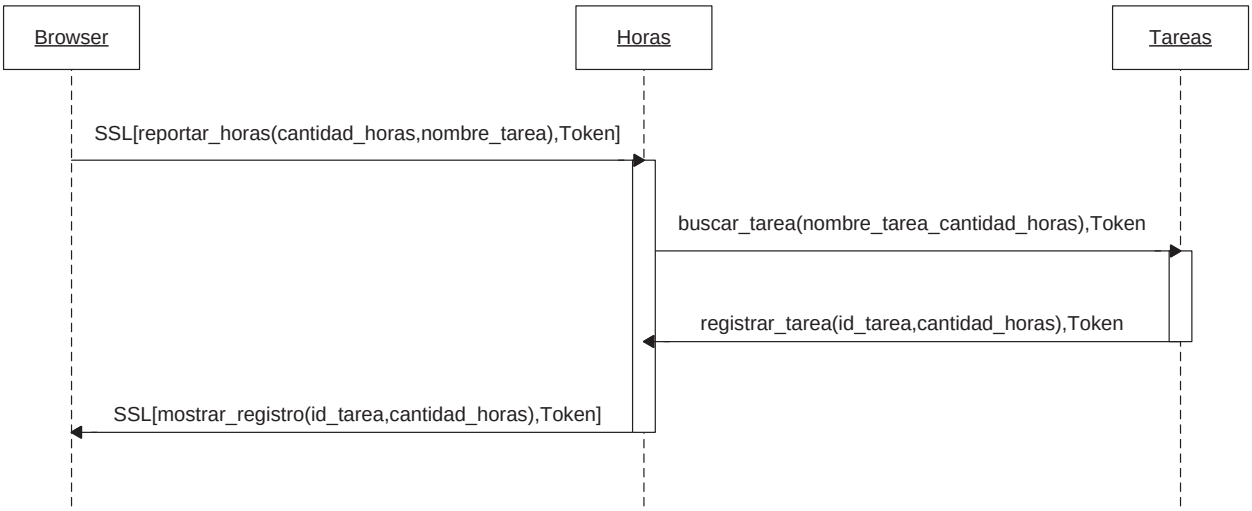

Figura 7.17 Diagrama de secuencia para el reporte de horas.

7.4.3 DIAGRAMA DE COMPONENTES

A continuación se presenta el diagrama de componentes del problema presentado, el cual se puede ver en la Figura 7.18.

En él se puede ver que se tienen 4 módulos que interactúan directamente con el usuario, es decir, tienen una interfaz visible desde el browser del usuario, los cuales están representados por un borde más grueso. Es útil conocer esta particularidad debido a que estos módulos necesitarán un mayor nivel de validación en los datos que reciben como entrada para sus procesos, ya que los reciben directamente desde el browser.

Todos los módulos se comunican entre sí mediante SSL, a excepción de la comunicación con el módulo que accede a la base de datos, ya que ésta es una comunicación interna entre los servidores, sin que sea accesible desde la red pública directamente.

Se puede ver está identificada la forma en que se comunicarán los módulos, por ejemplo:

- El acceso a la base de datos es mediante JDBC.
- A excepción del módulo de acceso a base de datos, el resto de los módulos se comunica mediante SSL.

- A excepción del módulo de acceso a base de datos, el resto de los módulos se intercambia un ID de sesión cada vez que se comunican.

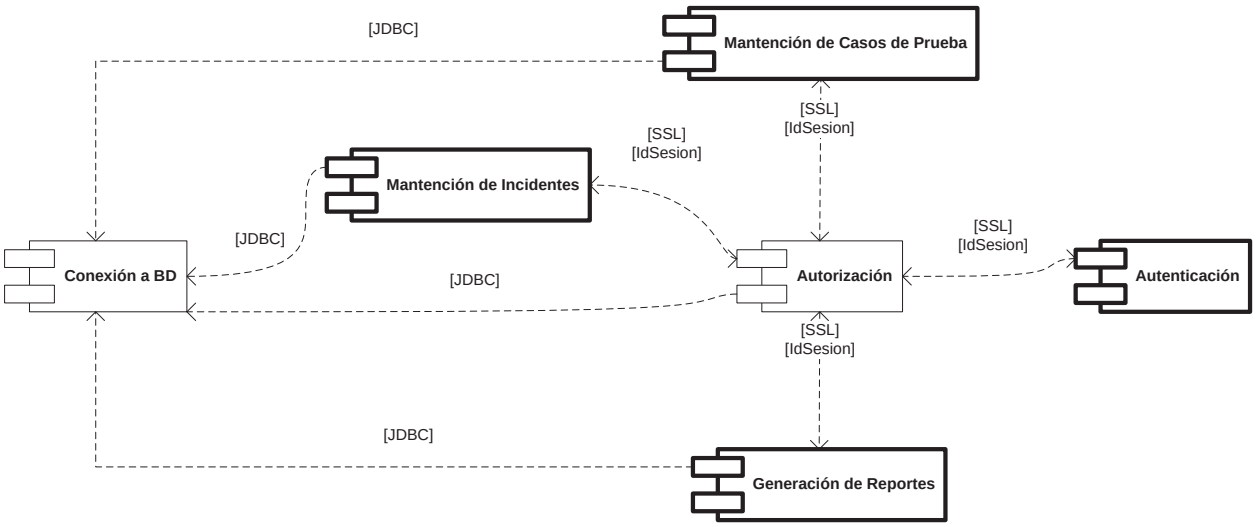

Figura 7.18 Diagrama de componentes.

7.4.4 DIAGRAMA DE CLASES

A continuación se presenta el diagrama de clases del problema presentando, el cual se puede ver en la Figura 7.19.

En este se puede ver que en para los usuarios la clave se debe almacenar utilizando un algoritmo de *hash*, en este caso SHA-256, de forma de que esta no sea visible, a pesar de que se tenga acceso a la base de datos.

Además, en cada clase, se puede ver que se detalla la cantidad estimada de datos o registros que manejará, por lo que se pueden tomar las medidas necesarias para que la aplicación pueda manejarlas correctamente. Así, se podrán evitar problemas de rendimiento que afectarán la disponibilidad de la aplicación desarrollada.

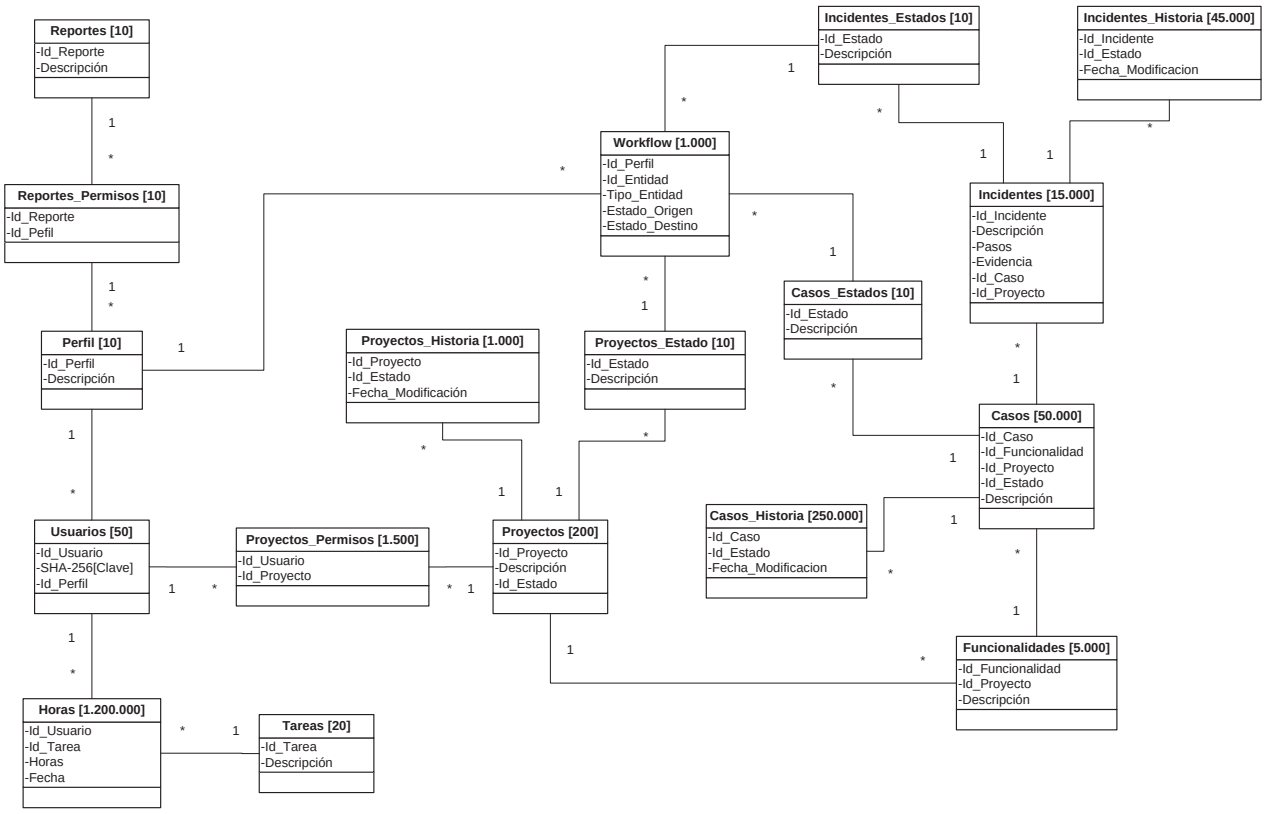

Figura 7.19 Diagrama de clases.

7.4.5 DIAGRAMA DE ACTIVIDAD

A continuación se presentan los diagramas de actividad para el problema presentado, los cuales ayudarán a entender de mejor manera las transiciones de las entidades que utilizan este concepto y que roles pueden realizar estos cambios. Dichos roles están representados en las etiquetas que se utilizan en las líneas que unen los diferentes estados posibles, de forma de que sea fácil saber quién debe realizar esta tarea.

En la Figura 7.20 se puede ver el *workflow* de estados para los proyectos que ingresan a la aplicación para ser probados. En él se puede ver que, por ejemplo, el único perfil que puede volver atrás un proyecto es el "Jefe de Proyecto QA".

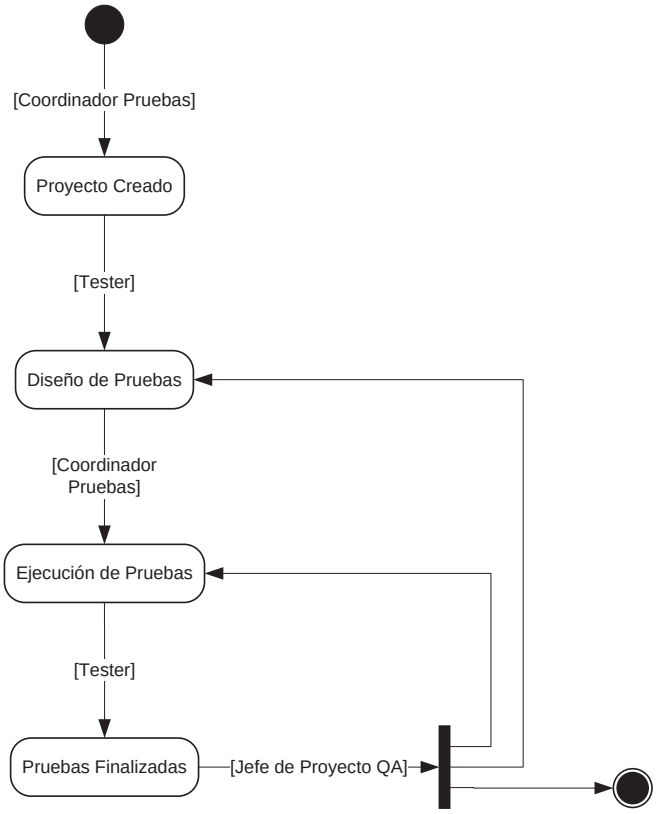

Figura 7.20 Diagrama de actividad de proyectos.

En la Figura 7.21 se puede ver el *workflow* de estados para los casos de prueba, en el cual se ve que el único perfil que interactúa con estos es el "Tester", ya que es este el que realiza todos los cambios de estado posibles.

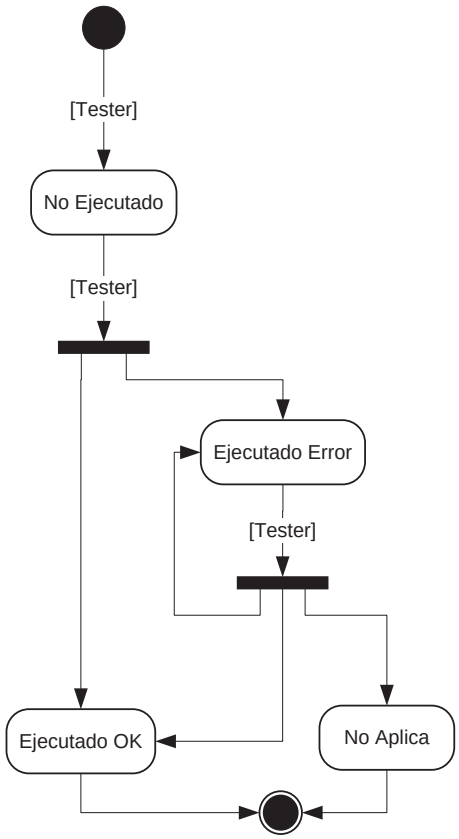

Figura 7.21 Diagrama de actividad de casos de prueba.

En la Figura 7.22 se puede ver el *workflow* para los incidentes que se generan durante la etapa de pruebas, en el cual, como se pude ver en la figura, interactúan más perfiles que en los otros casos. Ya que se tiene a los siguientes:

- Desarrollo
- Tester
- Coordinador Pruebas

Este es el ejemplo más claro de la importancia de tener un estricto control de los perfiles y sobre qué puede realizar cada uno de ellos, ya que la interacción la realizan perfiles o usuarios, que generalmente no están en un mismo lugar físico, por lo que una aplicación de este tipo puede ser de gran ayuda al momento de comenzar a corregir los errores o incidentes encontrados por el equipo de pruebas.

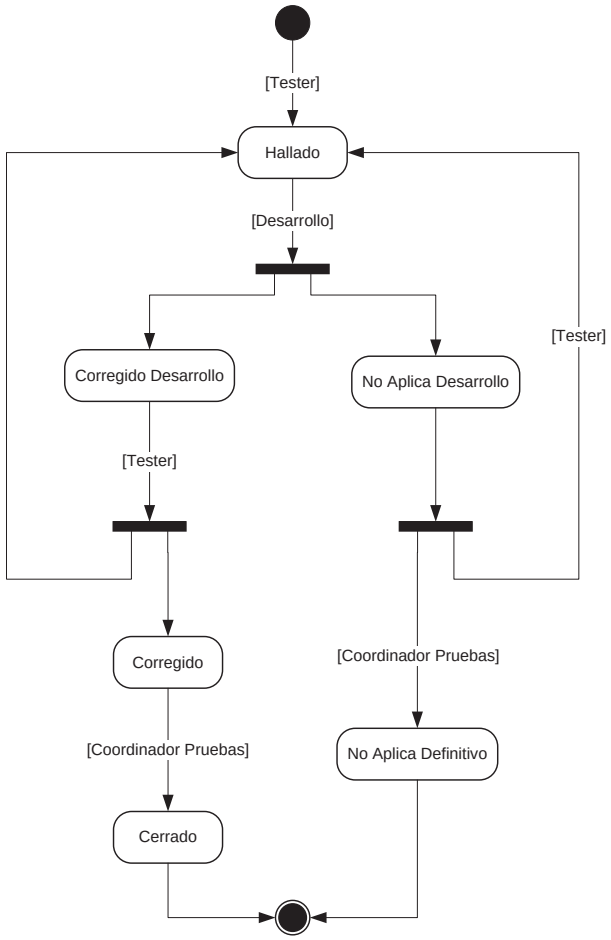

Figura 7.22 Diagrama de actividad de incidentes.

7.5 COMENTARIO SOBRE LOS RESULTADOS OBTENIDOS

Una vez aplicado el perfil propuesto al problema escogido para su comprobación, es posible extraer ciertas observaciones o comentarios útiles para su discusión.

De los resultados obtenidos de la aplicación del perfil se puede observar que los diagramas resultantes no difieren notoriamente de los diagramas estándares de UML, lo que trae los siguientes beneficios:

- No se necesita un esfuerzo mayor en entrenamiento para el equipo de desarrollo
- Cualquier persona con conocimientos de UML y de seguridad debiera ser capaz de comprenderlos

El hecho de que sea fácil de leer es una de las principales razones por las que se decidió presentar una alternativa a UMLSec, debido a que, como se comentó anteriormente, este perfil complica el UML original, necesitando gastar recursos para primero para comprenderlo, y luego para aplicarlo correctamente.

Además, esta propuesta es capaz de modelar un aspecto que UMLSec no considera, como lo es el de cantidad de datos a manejar. Si bien se podría decir que no es un aspecto estrictamente de seguridad, ya que no produce una vulnerabilidad que podría dar accesos no permitidos a un atacante, el hecho de no considerarlo puede producir que la aplicación falle o deje de responder ante una cantidad masiva de requerimientos o accesos. Como se explicó anteriormente, la denegación de servicio (producir que una aplicación web deje de operar) es considerada como un ataque de seguridad, por lo que es necesario evitar, dentro de lo posible, que se produzca.

La denegación de servicio, generalmente, es utilizada como último recursos por un atacante, ya que al no poder acceder de manera maliciosa a una aplicación para robar o destruir información, lo que hace es generar una gran cantidad de accesos a ésta para que el hardware o software colapsen y dejen de proveer el servicio. Esto puede producir diversos problemas, como:

- Afectar la imagen de la organización
- Costos por no poder utilizar una herramienta indispensable o por la que se cancela un valor
- Multas del estado o por contratos y compromisos adquiridos

Sin embargo, resta probar la propuesta del presente documento en sistemas o aplicaciones de mayor tamaño y complejidad, con el fin de verificar si es capaz de modelarlos correctamente sin dejar información importante fuera del análisis y diseño.

8 CONCLUSIONES

Toda organización que maneja datos sensibles y privados de los usuarios debe integrar a sus procesos internos algún estándar de seguridad que permita administrar de mejor manera estos datos. De todas formas, esto no basta por sí solo, ya que también es necesario considerar la seguridad en el proceso de desarrollo.

De no hacer esto, existen una serie de vulnerabilidades que un atacante puede explotar, las cuales pueden comprometer una serie de datos privados o la integridad del sistema mismo. El problema es que muchas organizaciones no consideran esto hasta que sufren algún ataque con consecuencias serias, como las que ya hemos conocido en el último tiempo en Chile y el resto del mundo (robos de bases de datos, clonación de tarjetas de crédito, etc.).

Con la integración de las pruebas de seguridad al ciclo de desarrollo se espera producir aplicaciones cada vez más seguras que logren hacer cada vez más difícil el acceso a la información. Teniendo en mente la seguridad desde el momento del análisis y diseño, se pueden lograr aplicaciones más robustas, permitiendo disminuir los riesgos de robos de datos privados y las correcciones que se deben hacer hacia el final del proceso de desarrollo.

Los perfiles UML son de gran ayuda en este sentido, ya que permiten comenzar a hablar de seguridad mucho antes en el proceso de desarrollo de lo que se hace actualmente, pero también deben ser lo suficientemente claros para que no agreguen un mayor grado de dificultad al entendimiento del lenguaje de modelado, contrarrestando las ventajas que se obtienen al manejar una mayor cantidad de información.

Por esta razón un perfil UML debe mantener la simpleza en los diagramas y en la notación, mientras se intenta agregar más información a estos. El objetivo es que los usuarios y los desarrolladores puedan seguir utilizando estas herramientas como lo han hecho hasta ahora sin tener que pasar por un período de adaptación muy largo o tener que estar consultando un manual cada vez que se quiera confeccionar o utilizar un diagrama.

Mientras antes se corrigen los errores en el proceso de desarrollo, más barato sale corregirlos y esto es cierto para cualquier tipo de requerimiento. Debido a esto, mientras más puntos de verificación se consideren, se disminuye la probabilidad de encontrar errores más avanzado en el proceso de desarrollo. Obviamente esto debe hacerse con ciertas precauciones, ya que se corre el riesgo de agregar demasiado trabajo o una mayor dificultad en la compresión del análisis y diseño, lo que puede provocar que el trabajo tome más tiempo o que no se logren los objetivos deseados.

Como se puede ver en los diagramas resultantes de la solución al problema planteado, utilizando la propuesta de perfil UML, no se modifica la estructura de los diagramas, sólo agrega información relevante que es necesario discutir con detalle al momento del diseño de la solución, como lo es la forma de autenticación de los usuarios, la forma de comunicarse del usuario con la aplicación, los algoritmos criptográficos a utilizar o la cantidad de datos que se espera el sistema administre.

Los diagramas obtenidos con la propuesta de perfil entregan más información a los desarrolladores, sin modificar la base de estos, lo que permite establecer de manera más clara como se quiere que la aplicación opere, permitiendo que dudas sobre si usar SSL o no y qué módulos lo utilizarán, no surjan a última hora, sino que estén claras desde un principio, ayudando a disminuir los tiempos de desarrollo, bajando la cantidad de correcciones que deben hacerse al diseño. Todo esto es posible sin tener que incurrir en un período de aprendizaje muy grande, ya que la estructura básica de los diagramas es la misma.

Mientras más acotado y definido se encuentre el diseño del software, mejor será el trabajo del equipo de desarrollo, ya que cualquier ambigüedad o mala definición genera dudas en los equipos, lo que puede provocar que se pierda tiempo en consultas para aclararlas o, lo que es peor, se asuman definiciones que después deberán ser cambiadas cuando la dirigencia del proyecto se dé cuenta, ya sea durante el desarrollo o las pruebas. Todas estas malas definiciones o simplemente ausencia de ellas provocará retrasos ya que, como se dijo anteriormente, mientras más tarde se encuentre un error, más caro saldrá corregirlo.

8.1 TRABAJO FUTURO

Para futuros trabajos se espera verificar la validez de la presente propuesta en un entorno de desarrollo completo y considerando la etapa de implementación. Esto con el fin de obtener retroalimentación por parte de los desarrolladores.

Además, se puede considerar la posibilidad de realizar pruebas de seguridad, con el objetivo de comprobar la calidad del producto final desarrollado.

9 REFERENCIAS

[OWASP, 2007] "The Ten Most Critical Web Applications Security Vulnerabilities", OWASP Fundation, 2007.

[OSA, 2007] "Open Security Architecture", http://www.opensecurityarchitecture.org, OSA

[TrustWave, 2008] "Secure Applications Development", Course Materials by Kevin Stadmeyer and Tom Leavey, TrustWave.

[Parker, 1998] "Fighting Computer Crime: A New Framework for Protecting Information", Donn B. Parker, Wiley 1998.

[Scambray, 2002] "Web Applications (Hacking Exposed)", Joel Scambray, Mike Shema., McGraw-Hill, 2002.

[PCI, 2008] "About the PCI DSS", www.pcisecuritystandards.org, Payment Card Industry Security Standard Council.

[IBM, 2003] "UML basics: An introduction to the Unified Modeling Language", www.ibm.com, 2003.

[OMG, 2005] "Introduction to OMG's Unified Modeling Language", www.omg.org, 2005.

[Fuentes, et al., 2004] "Una Introducción a los Perfiles UML", Lidia Fuentes y Antonio Vallecillo. Novática, 2004.

[Jürjens, 2002] "Using UMLsec and Goal Trees for Secure Systems Development", Jan Jürjens. Computing Laboratory, University of Oxford, GB and Software & Systems Engineering, Dep. of Informatics, TU München, 2002.

[Soldal, et al., 2003] "UML Profile For Security Assesstment", Mass Soldal Lund, Ida Hogganvik, Fredrik Seehusen, Ketil, Stolen. SINTEF Telecom and Informatics, 2003.

[Menezes, et al., 1996] "Handbook of Applied Cryptography", A. Menezes, P. van Oorschot, S. Vanstone. CRC Press, 1996.

[Daley, et al., 1999] "Data Encryption Standard", William M. Daley, Raymond G. Kammer. Federal Information Processing Standards Publication, 1999.

[EFF, 1998] "EFF DES Cracker Machine Brings Honesty to Crypto Debate", http://www.eff.org, 1998.

[FIPS 197, 2001] "Advanced Encryption Standard", Federal Information Processing Standards Publication 197, 2001.

[Schneier, 1994]	"Description of a New Variable-Length Key, 64-Bit Block Cipher", Brice Schneier. Fast Software Encryption, Cambridge Security Workshop Proceedings. Springer-Verlag, 1994.
[Wang, et al., 2008]	"How to Break MD5 and Other Hash Functions", Xiaoyun Wang and Hongbo Yu. Shandong University, Jinan 250100, China, 2008.
[FIPS 180-2, 2002]	"Secure Hash Standard", Federal Information Processing Standards Publication 180-2, 2002.
[Schneier]	"Cryptanalysis of SHA-1", Brice Schneier. http://www.schneier.com/blog/.
[RFC5246, 2008]	"The Transport Layer Security (TLS) Protocol Version 1.2", T. Dierks, E. Rescorla. RFC5246, 2008
[IT Governance Institute, 2007]	"COBIT 4.1, IT Governance Institute. 2007
[ISO, 2005]	"ISO/IEC 17799:2005", International Organization for Standardization, 2005
[OWASP, 2008]	"OWASP Testing Guide v3", OWASP Fundation, 2008
[NIST, 2002]	"Risk Management Guide for Information Technology Systems". NIST, 2002