Elicitación y Trazabilidad de Requerimientos utilizando Patrones de Seguridad

Miguel A.Solinas

Director: Dr. Eduardo Fernández

Sub-Director: Mg. Leandro Antonelli

Tesis presentada a la Facultad de Informática de la Universidad Nacional de La Plata como parte de los requisitos para la obtención del título de Magister en Ingeniería de Software.

La Plata, Marzo 2012
Facultad de Informática
Universidad Nacional de La Plata
Argentina

Índice General

Índice de Figuras

Índice de Tablas

1 Introducción y motivación

El desarrollo de aplicaciones seguras se ha transformado, en el comienzo de este siglo, en un tema de interés para el mercado mundial de desarrollo de software. Esto ha sido impulsado por varios motivos. En el exterior, particularmente en USA, se han propuesto en el año 2002, regulaciones como [HIPAA 2011], "Sarbanes–Oxley Act" [SOx 2011] y "Federal Information Security Management Act" [FISMA 2011]. La primera de ellas impulsa Reglas de Privacidad y Seguridad en el tratamiento de la información médica. Puntualmente sobre planes de atención médica civil y militar, oficinas de compensación médica y proveedores médicos que realicen transacciones financieras y administrativas por medios electrónicos. La segunda reconoce el impacto de la seguridad en el tratamiento de la información, sobre los intereses económicos del estado nacional, proponiendo que cada agencia federal deba desarrollar, documentar e implementar un programa para garantizar la seguridad en el tratamiento de la información.

A nivel global el interés en el desarrollo de aplicaciones seguras ha ido creciendo, gracias a la omnipresencia de internet que ha puesto cara a cara a usuarios en posiciones geográficas extremas, lo que ha facilitado que las siempre novedosas técnicas utilizadas por hackers, les permitan ganar acceso a datos sensibles, deshabilitar aplicaciones y realizar actividades maliciosas, ayudados en gran medida por las vulnerabilidades de los sistemas de software de los propios damnificados.

Estas vulnerabilidades del software producto de diseños defectuosos son las que tienden a agravar el problema. Permiten que una aplicación sea atacada con éxito a causa de una falla de seguridad. Luego la aplicación vulnerada permite propagar el ataque a través de la red o abrir el camino a nuevos ataques, produciendo daños en otras aplicaciones. Por otro lado, estos procesos son invisibles a los usuarios legos, si bien los expertos reconocen que las amenazas siguen creciendo.

Las vulnerabilidades introducidas por error o por malas prácticas en el software son un problema serio, que en el futuro pueden exponer a Empresas y organismos del Estado a situaciones muy difíciles y costosas de resolver.

Todas ellas son razones muy fuertes que hacen imperativo tomar acciones preventivas frente al desarrollo de software seguro, investigar sus fundamentos, minimizar las amenazas existentes y establecer una base de conocimiento y aptitudes que ayuden a los profesionales de la seguridad a reducir los riesgos en el mediano plazo.

Hasta hace no mucho tiempo atrás, el aspecto seguridad en la construcción de sistemas de software, era una tarea planificada "*a posteriori*" de la implementación completa del sistema. Los equipos de desarrollo se enfocaban exclusivamente en los requerimientos funcionales y esperaban atacar la seguridad una vez finalizada la tarea de implementación de estos requerimientos. Esta aproximación al desarrollo de una aplicación segura ha probado ser ineficaz. Abrió la puerta a vulnerabilidades no detectadas previamente, permitiendo que los sistemas sean atacados y dañados.

Todo ello, ha llevado a que diferentes grupos de investigación intenten responder las siguientes preguntas:

- ¿ Cómo producir software seguro?
- ¿ Cómo garantizar que se ha producido un software seguro?
- ¿ Cuáles son los vectores de transmisión de la seguridad dentro de un software?

Por otro lado, existen causas esenciales que producen vulnerabilidades y debilidades en el software [Ozmet 2006]. Una de las más relevantes es la ausencia de tecnología apropiada: herramientas y métodos para asistir en el desarrollo y producción de software seguro o determinar si el software producido por un equipo es más o menos seguro. Otra razón esencial, que impacta

directamente sobre la seguridad, es la regla del mercado que premia al que llega primero con un producto novedoso, no el más seguro, sin detenerse demasiado sobre los riesgos que pueda ocasionar sobre la organización. Esto también impacta directamente sobre la falta de motivación de los programadores para generar código seguro. Todo esto produce una mezcla compleja de gestionar.

De todos modos, existen criterios que pueden ayudar a generar software más seguro:

i) Nunca olvidar que la seguridad y costo de producción de un sistema de software depende fuertemente del conocimiento que se tenga de sus requerimientos [Garvin 1984].

ii) La incorporación del tratamiento de la seguridad en cada una de las diferentes etapas del ciclo de desarrollo de software es un criterio aceptado para mejorar la seguridad del producto final.

iii) Los patrones de seguridad representan las mejores prácticas logradas por la industria a fin de detener o limitar ataques a la seguridad [Schumacher 2006].

De esto podemos inferir que promover el uso de patrones de seguridad para que guíen y conduzcan en todo momento, la construcción de modelos de desarrollo de software seguro, partiendo de etapas tempranas del proceso, es un criterio que nos garantizará una mayor seguridad en el comportamiento de un producto de software.

En este trabajo se da una respuesta a las preguntas arriba planteadas. La propuesta tiene que ver con lograr que la seguridad sea un aspecto a considerar durante todo el proceso de desarrollo de software y muy en particular con etapas muy tempranas, concretamente: elicitación y análisis de requerimientos y diseño. En dichas etapas se deben incorporar las mejores prácticas para prevenir vulnerabilidades y fallas en el uso del software. Se deben identificar las amenazas. Se deben identificar elementos que actúen como vectores transmisores de la seguridad hacia el interior de un sistema de software. Estos elementos serán los que contrarresten las amenazas y conduzcan al software, desde la etapa de diseño hacia de la etapa de implementación. En este trabajo, estos elementos son los patrones de seguridad. Ellos son la sustanciación de las mejores prácticas conocidas y son los vectores conductores de la seguridad hacia el interior de un diseño.

Experimenté con LEL y Escenarios y descubrí que son elementos adecuados para modelar [Solinas 2009] comprender y estudiar Patrones de Seguridad. Leonardi [Leonardi 2001] y Antonelli [Antonelli 2003] han demostrado la eficacia de su uso en el estudio de otros dominios.

Descubrí que es posible modelar un Patrón de Seguridad mediante un sub-escenario (un episodio) de un escenario de una aplicación con requerimientos de seguridad. Tanto el sub-escenario como el escenario describen la dinámica de un caso de uso. Y en particular como el Patrón de Seguridad, ya tiene un comportamiento estático y dinámico conocidos, facilita su incorporación al modelo [Solinas 2009].

El Léxico Extendido del lenguaje (LEL) es un modelo contextual que permite capturar el lenguaje de un dominio. Identificando y definiendo los símbolos propios de un contexto se logra un mejor entendimiento de éste. La idea bajo el LEL es muy simple: primero hay que preocuparse por entender el lenguaje del problema, sin preocuparse por entender el problema. El LEL es un glosario que permite registrar signos (palabras o frases) los cuales son específicos del dominio. Cada signo del LEL es descripto de dos formas: a través de la noción y los impactos. Por otro lado, los aspectos dinámicos del modelo del contexto se capturan mediante escenarios. Para la construcción de cada escenario se toma como guía, a fin de considerar y organizar los requerimientos, un diagrama de casos de uso. Los casos de uso sitúan los requerimientos en el contexto de historias y objetivos de uso y nos permiten mostrar claramente los límites de la aplicación. Por último, a partir de LEL y escenarios es

posible obtener tarjetas CRC (colaboraciones y responsabilidades de las clases). Estas tarjetas identifican posibles clases para diseñar una solución orientada a objetos. Cada tarjeta representa un objeto del mundo real. Las CRC deben su nombre a que cada tarjeta identifica una clase. En ella se identifican las acciones que realiza: sus responsabilidades. Las tarjetas CRC también registran las relaciones con otras clases: sus colaboraciones.

En este trabajo se describe un método para incorporar Patrones de Seguridad en el dominio específico de una aplicación con requerimientos de seguridad. Se utiliza la relación entre un glosario de términos específicos (LEL) y los Escenarios, asociados a casos de uso, para abstraer el conocimiento de ambos dominios: el propio de la aplicación y el de la seguridad. Luego con una nueva versión de la herramienta Baseline Mentor Workbench (BMW), se puede deducir el conjunto de CRC de análisis: clases candidatas. La terna: LEL, Escenario y CRC, son los componentes del modelo que nos permiten representar un Patrón de Seguridad mediante un sub-escenario con su terminología, semántica y comportamiento específicos. La información documentada en el "*template*" del Patrón de Seguridad, concretamente su comportamiento estático y dinámico, colaboran para conducir y validar el análisis del dominio de la aplicación con requerimientos de seguridad. De este modo el Patrón de Seguridad no sólo se incorpora al modelo de la aplicación de forma natural y controlada, sino que conduce las decisiones a tomar sobre el diseño de la aplicación. Como toda la información sobre el modelo estático y dinámico del Patrón de Seguridad pertenece a la etapa de diseño, y se integra a un modelo contextual de una aplicación perteneciente a la etapa de elicitación y análisis de requerimientos, los elementos del Patrón de Seguridad se dice que traccionan hacia etapas más avanzadas en el proceso de desarrollo de la aplicación. La herramienta extiende el concepto de "*forward traceability*" sobre los requerimientos de seguridad y permite tener visibilidad entre las CRC y los requerimientos que le dieron origen, sean éstos requerimientos funcionales y de seguridad.

Este trabajo se ha organizado de la siguiente manera: en el Capítulo 2 se presentan aspectos relevantes del estado del arte de la producción de software seguro; se discuten algunos temas que apuntan a desarrollar un léxico y marco conceptual apropiado y otros que son motivos de investigación y desarrollo; se revisan los adelantos hechos sobre la elicitación de requerimientos de seguridad. En el Capítulo 3 se presenta el conjunto de elementos utilizados en la descripción del método propuesto: LEL, Escenarios, CRC y Patrones de Seguridad. En este Capítulo se muestra que es factible modelar un Patrón de Seguridad con los elementos descriptos: LEL, Escenario y CRC; lo cual resulta en un valor fundamental para este trabajo. En el Capítulo 4 se describe el método de elicitación de requerimientos utilizando Patrones de Seguridad, primero una síntesis de sus etapas y luego su aplicación en la construcción de una aplicación con requerimientos de seguridad. En el Capítulo 5 se describe la herramienta construida para utilizar el método propuesto. En el Capítulo 6 se presentan métodos relacionados, complementarios al propuesto en este trabajo. En el Capítulo 7 se presentan las conclusiones de este trabajo y para finalizar, en el Capítulo 8 se mencionan posibles futuros trabajos para continuar esta línea de investigación.

2 Estado del arte de la construcción de software seguro

Hasta finales del siglo pasado, cuando se hablaba de asegurar el software, se estaba hablando de dos propiedades puntuales: calidad y confiabilidad. En realidad se hablaba de asegurar el software como una forma abreviada de garantizar la calidad del mismo. Pasados los primeros años del nuevo siglo, la frase "asegurar el software" fue adoptada para expresar la idea de garantizar la seguridad del software, comparable al aseguramiento en el tratamiento de la información que se expresa como "aseguramiento de la información". A partir de ese momento, esta disciplina se define de muchas maneras, la mayoría de las cuales se complementan entre sí para diferir ligeramente en el énfasis sobre los términos que utilizan y el enfoque dado sobre el problema de construir software seguro.

En todos los casos, todas las definiciones de aseguramiento del software intentan transmitir la idea de que para garantizar la seguridad del software se debe proporcionar un nivel razonable de confianza de que el mismo funcionará correctamente, de forma predecible, en consonancia con los requerimientos documentados, y que su funcionalidad no podrá estar comprometida por un ataque directo o mediante el sabotaje intencional. Podríamos entonces aproximar una primera definición sobre qué significa garantizar la seguridad del software:

> *"Tomar las acciones que permitan obtener un nivel justificable de confianza de que el software mostrará siempre todas las propiedades necesarias para garantizar su funcionamiento, de acuerdo a los requerimientos documentados, y continuará operando de forma fiable a pesar de la presencia de fallas intencionales. En la práctica, dicho software debe ser capaz de resistir la mayor cantidad de ataques y tolerar el mayor número posible de aquellos que no puede resistir; conteniendo el daño y recuperándose a un nivel normal de operación tan pronto como sea posible, pasados aquellos ataques que no es capaz de resistir o tolerar."*

La NASA asocia el concepto de "*garantizar la seguridad del software*", a un conjunto planificado y sistemático de actividades que garanticen que los procesos y productos del software se ajustan a los requerimientos, normas y procedimientos. Se enfoca sobre un "*conjunto planificado y sistemático de actividades*", que incluyen 1) Especificación de requerimientos, 2) Pruebas, 3) Validación y 4) Informe formal de resultados; llevadas a cabo a lo largo del ciclo de vida del desarrollo de software.

De acuerdo con "*The Information Assurance Technology Analysis Center*" (IATAC), dependiente del "*The Defense Technical Information Center*" (DTIC), un software es seguro si es robusto frente a ataques. Esto significa que un software seguro sigue siendo fiable, incluso cuando su fiabilidad se ve amenazada. Un Software seguro no puede ser corrompido ni saboteado. En término prácticos, esto se traduce en que no debería presentar fallas o debilidades que puedan ser utilizadas por atacantes humanos o por código malicioso.

Hay definiciones como la propuesta por el "*U.S. Department of Homeland Security*" (US DHS) que asocian fuertemente el concepto de software seguro con aquel producto que presenta ausencia total de vulnerabilidades explotables por código malicioso o mal intencionado. Esto trae aparejado inmediatamente la necesidad de poder determinar con total certeza tal ausencia. Acá se plantea el problema de que la seguridad, a diferencia de otras propiedades del software, aún no está plenamente comprendida como para que a lo largo de todo el ciclo de vida del software, se pueda determinar con total certeza que el software sea seguro; o al menos en qué medida se lo puede considerar seguro.

Por otro lado, hay que tener en cuenta que la seguridad del software es una propiedad dinámica: aquello que hace que sea seguro en un ambiente en particular o en un contexto de amenazas

puede llevarlo a no hacerlo seguro en otro ambiente o en otro contexto de amenazas diferentes, o si el software cambia en su funcionalidad. Incluso en términos de testeabilidad, la seguridad también presenta problemas para su valoración. Se pueden ejecutar miles de horas de prueba y al finalizar, afirmar que el software que pasó las pruebas operará de manera confiable, pero lo mismo no se puede decir respecto a la seguridad. Lamentablemente las técnicas para poner a prueba la seguridad del software son inmaduras y colectivamente representan un mosaico incompleto para el amplio espectro de problemas que deberían someterse a ensayo.

De todos modos, la disciplina que enfoca la seguridad del software ha evolucionado rápidamente en los últimos 10 años y la cantidad de iniciativas, normas, recursos, metodologías, herramientas y técnicas disponibles, orientadas a entender y mitigar los problemas de seguridad han aumentado exponencialmente.

Integrando los conceptos arriba expuestos, se puede decir que un software es seguro cuando da cumplimiento a los siguientes aspectos:

a) No puede ser forzado intencionalmente a fracasar en el cumplimiento de sus requerimientos funcionales y continuará siendo fiable y adecuado para la tarea para la que se lo construyó a pesar de los esfuerzos intencionales por comprometer su funcionamiento.
b) Debe ser diseñado, implementado y configurado para seguir funcionando correctamente ante la presencia de la mayoría de los ataques, fallas o debilidades conocidas en el software, tolerando los errores y fracasos que resulten de tales ataques, fallas o debilidades.
c) Debe ser diseñado, implementado y configurado para aislar, contener y limitar los daños ocasionados por fallas o defectos causados por los ataques que el software fue incapaz de resistir o tolerar y recuperarse lo más rápidamente posible a partir de los fracasos.

Para lograr estos objetivos, deben existir evidencias de que ha sido diseñado, implementado y configurado de manera tal que:

a) Las fallas y debilidades explotables han sido evitadas primero por los diseñadores y luego por los desarrolladores del software.
b) Los desarrolladores mal intencionados han visto reducidas o eliminadas las probabilidades de implantar intencionalmente fallas o debilidades explotables en la lógica del software.
c) Se han utilizado en el diseño las mejores prácticas que garanticen que el software sea resistente, tolerante y fuerte frente a ataques.
d) Las interacciones entre los componentes dentro del sistema de software y entre entidades del sistema y entidades externas, no contienen debilidades explotables.

A continuación, a fin de dar una visión global del estado del arte de la producción de software seguro, orientados con el interés de este trabajo, se presentan aspectos relevantes, presentando temas que apuntan a desarrollar una terminología y marco conceptual apropiado y otros que son motivos de investigación y desarrollo.

2.1 Software seguro vs aplicación segura. Calidad, confiabilidad y seguridad.

Internet plantea una infraestructura de comunicaciones distribuida, y sobre ella se han montado múltiples sistemas de software que han mostrado ser vulnerables. Esto ha llevado a un gradual reconocimiento para aumentar las medidas que garanticen la seguridad de dicha infraestructura: la red, lo cual ha demostrado no ser suficiente. Es necesario tomar medidas adicionales

que garanticen de alguna manera la seguridad del software. De ello surge que cuando se habla de garantizar la seguridad de una aplicación, el concepto incluye medidas para garantizar la seguridad del software y la infraestructura de red.

Garantizar la seguridad de la aplicación combina técnicas de ingeniería de sistemas, tales como defensas en profundidad, cortafuegos de capa de aplicación, gateways de seguridad XML, cajas de arena o firma de código; configuraciones seguras y prácticas de seguridad operativa, incluyendo gestión y administración de parches para vulnerabilidades.

Las medidas para una defensa en profundidad, incluyen técnicas que operan mediante el uso de protecciones sobre los límites de la aplicación a fin de reconocer y bloquear patrones de ataque; el uso restringido en entornos de ejecución para aislar aplicaciones vulnerables, reduciendo así al mínimo su exposición a ataques y posible interacción con componentes menos fiables.

Lo referente a la seguridad operativa incluye medidas que se enfocan en reducir el número y exposición de vulnerabilidades de las aplicaciones (–Ej. a través de parches–) revalorando ocasionalmente la gravedad de las vulnerabilidades residuales y las amenazas que puedan ser objetivos a explotar, de modo de ajustar en consecuencia la defensa en profundidad a fin de mantener el nivel requerido de eficacia de la aplicación.

En este aspecto, la tarea de asegurar una aplicación proporciona una base adecuada para el aseguramiento de software, comparable a las garantías de calidad y confiabilidad que se puede lograr, mediante el uso de prácticas establecidas por el control de calidad y técnicas de tolerancia a fallos, respectivamente. En este punto, debemos marcar las diferencias entre las metas que persiguen características como calidad, confiabilidad y seguridad.

Para ello tenemos que plantear la siguiente paradoja: si bien los profesionales que atienden la seguridad de una aplicación reconocen que el cómo ha sido diseñada y construida una aplicación, es significativo en términos de reducción de la probabilidad y número de vulnerabilidades que contiene la aplicación, estos profesionales pueden no estar directamente involucrados con su proceso de desarrollo. Por otro lado, un software que pretenda ser seguro requiere que la seguridad sea vista como una característica esencial, una propiedad que estará mejor garantizada si se precisa desde el principio del proceso de desarrollo del software. De modo que la seguridad del software debe ser abordada de forma global y sistemática; de la misma manera que se hace con la calidad del software, pero por parte de los profesionales específicos que participan en su construcción.

Cabe la pregunta: ¿Qué diferencias hay entre seguridad, calidad, y confiabilidad del software? Por lo pronto las diferencias no están en los objetivos que se persiguen. Todas estas características impulsan acciones para garantizar que el software sea confiable a pesar de la presencia de determinados estímulos, influencias y circunstancias de origen interno y externo. La diferencia radica en la naturaleza de aquellos estímulos, influencias y circunstancias que pretenden cada una eliminar o atenuar.

Estas diferencias se pueden caracterizar en términos de amenazas sobre una propiedad que se desea alcanzar, ya sea calidad, confiabilidad o seguridad. La principal amenaza para la calidad es de origen interno: presencia en el propio software de fallas y defectos que amenazan su capacidad de funcionar de manera previsible, de acuerdo a sus requerimientos. Debido a que tales defectos y sus resultados se deben a errores de valoración o ejecución por parte del desarrollador del software al probarlo, instalarlo u operarlo, se concluye que a menudo estas amenazas son NO intencionales.

Si miramos las principales amenazas que ponen en riesgo metas de confiabilidad, concluimos que son de origen interno y externo. Son siempre fallas accidentales, en donde no hay ninguna

intencionalidad de explotarlas. Incluyen amenazas sobre la calidad, amplificadas y aumentadas por amenazas surgidas de la ejecución del software en un medio ambiente que en determinados momentos, se comporta de forma impredecible, por la razón que sea.

Pero las fallas que sin intencionalidad, accidentalmente, ponen en peligro la calidad y confiabilidad del software también pueden ser intencionalmente explotadas por agentes humanos (atacantes) o agentes maliciosos de software (software malicioso). Y aquí está la diferencia: las amenazas que ponen en riesgo la seguridad del software se diferencian de aquellas que ponen en riesgo la confiabilidad y la calidad por su intencionalidad. Las amenazas que ponen en riesgo la seguridad del software son intencionales, no son accidentales.

Pero la presencia de aquellas vulnerabilidades que permiten a las amenazas de seguridad lograr sus objetivos, no pueden ser intencionales. Sí es intencional la explotación focalizada de esas vulnerabilidades. John Pierce McDermott del Naval Research Laboratory's Center for High Assurance Computer Systems (NRL CHACS) [McDermott 2005] presenta la diferencia entre las amenazas a la seguridad, por un lado y las amenazas a la robustez, confiabilidad, calidad, etc, por el otro. Caracteriza las amenazas en términos de fallas estocásticas (no intencional) vs fallas patrocinadas (intencionales). Las fallas estocásticas pueden causar defectos de software para convertirse en vulnerables a ataques, pero a diferencia de las fallas patrocinadas su existencia no es intencional.

2.2 Ingeniería de los Sistemas Seguros

La Ingeniería en Sistemas tiene a su disposición diferentes modelos de proceso para utilizar en sus desarrollos, tales como el modelo en cascada y espiral. Teniendo en cuenta que para la ingeniería de los sistemas seguros es necesario un enfoque centrado en los riesgos, un modelo de desarrollo en espiral es ideal, ya que es un enfoque impulsado por los riesgos. Sin embargo, el modelo de proceso en "V", concebido por Forsberg y Mooz, [Forsberg 1991] es un medio que a los fines ilustrativos, ayuda a comprender las relaciones de los sistemas, con la ingeniería de software, las pruebas y la validación. La Figura 2.1 se deriva de este modelo en V, para mostrar la relación entre la ingeniería de sistemas seguros y la ingeniería de software seguro.

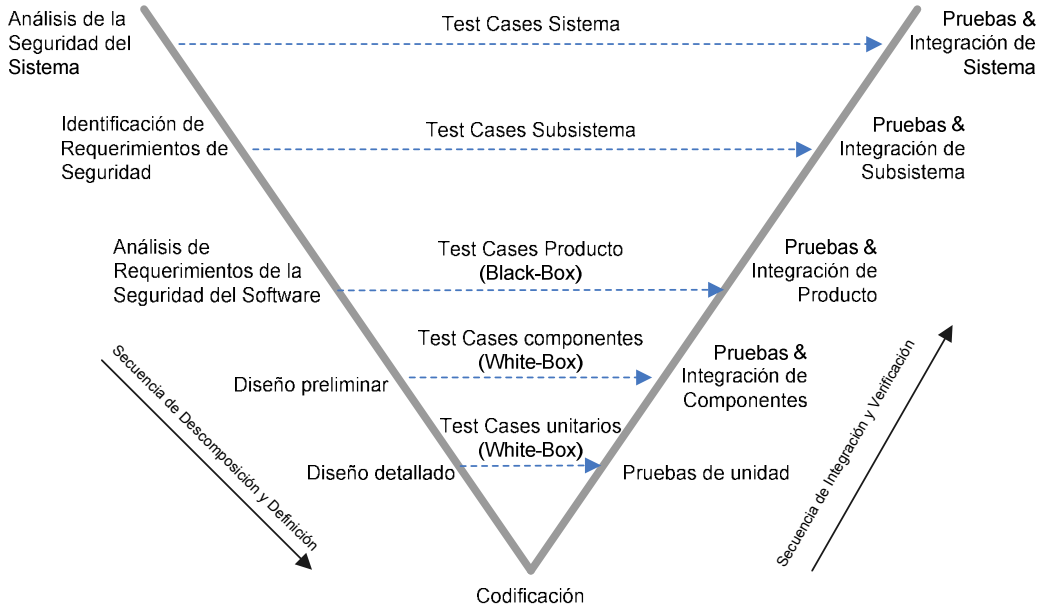

Figura 2.1: Modelo de proceso en V aplicado a la Ingeniería de los Sistemas Seguros.

En el modelo en V, el lado izquierdo hacia abajo representa las principales fases técnicas que participan en la transformación de los requerimientos de seguridad de un sistema en funcionalidades de un software. El lado derecho hacia arriba representa las fases técnicas de prueba, integración y validación asociados a cada fase en el lado izquierdo. Las flechas punteadas muestran cómo los artefactos de la descomposición y las fases de definición proporcionan información a los casos de prueba utilizados en las fases de integración y verificación.

La fase de análisis de la seguridad del sistema incluye el análisis de los requerimientos del sistema, el diseño y el desarrollo. En relación al diseño de sistemas, los Ingenieros en Sistemas son los responsables del diseño de una solución total del sistema. Para diseñar la seguridad del sistema, hay disponibles opciones de hardware y de software, incluyendo las mencionadas en el punto anterior y que caracterizan a una aplicación, que es la infraestructura de comunicaciones. El objetivo es realizar una compensación consciente de los riesgos para lograr soluciones adecuadas que permitan construir un sistema seguro. Esto se realiza a un nivel amplio, en bloques jerárquicos, como los considerados en la construcción de una arquitectura.

Durante la fase de asignación de requerimientos de seguridad el objetivo es asignar los requerimientos del sistema a los componentes de hardware, software, etc. En cuanto a las consideraciones respecto al software durante el diseño del sistema, un reciente informe de *"National Defense Industrial Association"* (NDIA), "*Top Software Engineering Issues within the Department of Defense and Defense Industry*" [NDIA 2010], observa que en muchas organizaciones, las decisiones referidas a la ingeniería de sistemas se hacen sin la participación plena de Ingenieros de Software. Esto, potencialmente, puede dar lugar a soluciones de sistemas que desde una perspectiva de la seguridad del software, no responden adecuadamente a las fallas que causan las vulnerabilidades de seguridad en el diseño del software resultante.

Los requerimientos asignados al software posteriormente son entregados a los ingenieros de software y los requerimientos de hardware a los responsables de la ingeniería de hardware, quienes llevan a cavo un análisis de requerimientos de software o hardware más detallado, un posterior diseño, etc., que representan en el diagrama "V", las fases de Análisis de Requerimientos de Seguridad del software hasta el Diseño Detallado, codificación e integración de productos y pruebas. El ingeniero en sistemas posteriormente recibe los componentes terminados, tanto de software como de hardware y es responsable de la integración de los subsistemas y prueba final de integración y aceptación.

En conclusión, los requerimientos de seguridad del software, en la construcción de un sistema, son una parte fundamental para garantizar una correcta solución. Sin embargo los "*stakeholders*" (en particular los usuarios de los sistemas) no pueden ser totalmente conscientes de los riesgos que representa la no seguridad para los cumplimientos de la misión de la organización, ni tampoco de los riesgos de las vulnerabilidades asociadas a sus sistemas. Es por ello que para definir los requerimientos de seguridad, es recomendable que los ingenieros en sistemas, junto a los usuarios, puedan realizar un análisis top-down y bottom-up de las fallas de seguridad que podrían poner en riesgo a la organización, así como definir las acciones para enfrentar las vulnerabilidades.

El análisis de árboles de falla para la seguridad (a veces conocido como árbol de amenazas o análisis de árbol de ataque) es un enfoque top-down para la identificación de vulnerabilidades. En un árbol de fallas, el objetivo del atacante se coloca en la parte superior del árbol. A continuación, el analista documenta posibles alternativas para lograr la meta que atacante. Para cada alternativa, el analista puede añadir, de forma recursiva, las alternativas para el alcance de los objetivos secundarios que componen el objetivo principal del atacante. Este proceso se repite para cada meta del atacante.

Mediante el examen de los nodos de nivel más bajo del árbol de ataque resultante, el analista puede identificar las posibles técnicas para violar la seguridad del sistema. Las acciones para prevenir estas técnicas podrían formar parte de la especificación de requerimientos de seguridad del sistema.

El Análisis de Efectos y Modos de Falla (Failure Modes and Effects Analysis – FMFE) es un enfoque down-top, que permite analizar los posibles fallos de seguridad. Las consecuencias de un fallo simultáneo de todos los mecanismos de protección, existentes o previstos, están documentados, y se traza el impacto de cada fracaso en la misión del sistema y de los "*stakeholders*".

Otras técnicas para el desarrollo de requerimientos de seguridad de un sistema incluyen el modelado de amenazas y el modelado de casos de mal uso y abuso. Ambas técnicas se describen más adelante. Los requerimientos también pueden ser derivados de modelos de políticas de seguridad y objetivos de seguridad del sistema que describen los mecanismos necesarios del sistema de protección [por ejemplo, la descripciones de las Metas de Evaluación (Target of Evaluation – TOE) producida por las evaluaciones de Common Criteria (CC)].

El análisis de árboles de ataque y FMEA aumentan y complementan los requerimientos de seguridad derivados del modelo de amenazas del sistema, del modelo de políticas de seguridad, y/o objetivos de seguridad. Los resultados del análisis de requerimientos de seguridad del sistema se pueden utilizar como base para los escenarios de prueba de la seguridad que se utilizarán durante las pruebas de integración o aceptación.

Para ser efectivamente tratados durante la fase de integración, los problemas con la seguridad del sistema deben ser identificados en la etapa de requerimientos y contemplados en la fase de diseño y posteriores. En la construcción de aplicaciones, diferentes componentes del sistema interactúan con sistemas externos, cuyas características de seguridad pueden ser inciertas o dudosas si la seguridad de una aplicación descansa sobre el conocimiento del comportamiento e interacciones de sus componentes y subsistemas. Entonces, sobre la base de un análisis de riesgo de sus componentes, pueden determinarse los mecanismos de protección necesarios.

Como se menciona en el borrador sobre Seguridad en el Ciclo de Vida del Software, de DHS:

> *Para determinar la seguridad de un sistema que contiene un componente, módulo o programa dados, se requiere de un análisis de cómo se utiliza ese componente/módulo/programa en el sistema, y cómo actúa el sistema en su conjunto para mitigar o reducir el impacto de cualquier compromiso de un componente individual, lo cual pueda derivar en un ataque exitoso al componente o a las interfaces entre ellos. De ello surge que los riesgos de la no seguridad puede ser reducidos a través de:*
>
> - *Examinar todos los componentes adquiridos, reutilizados, y creados desde cero, antes de su aceptación e integración en el sistema;*
> - *Examinar las interfaces, observar las relaciones de confianza e implementar un encapsulado cuando sea necesario;*
> - *Testear la seguridad del sistema como un todo.*

Si se va a construir un artefacto de software desde cero, y se desea contemplar el aspecto seguridad, se deben integrar las actividades referidas a la seguridad en los procesos que hacen al ciclo de vida del desarrollo de software. Hay que tener en cuenta entonces, actividades como los test de penetración o diseños que cumplan con políticas de menor privilegio, sumadas a las actividades y principios ya llevados a cabo en cada una de las fases del ciclo de vida del desarrollo de software.

A pesar de lo dicho y de las actividades de investigación en este campo, Mouratidis and Giorgini [Mouratidis 2007] dicen que no existe aún una metodología de ingeniería de software que garantice la seguridad en el desarrollo de grandes sistemas de software. Pero algunas coincidencias aparecen, por ejemplo: sin importar el modelo de proceso que se vaya a utilizar, la seguridad debería ser considerada desde las etapas más tempranas del ciclo de vida de desarrollo del software. Por ello primero consideraremos aspectos que afectan transversalmente la totalidad del CVDS o al menos múltiples etapas, tales como la gestión de riesgos y el uso de métricas de seguridad; en el punto 2.3 nos enfocaremos en presentar los avances hechos sobre el estudio de la incorporación de la seguridad en la etapa de estudio de requerimientos, revisando los diferentes modelos de procesos para el CVDS en el punto 2.4.

2.3 Aspectos que impactan sobre la totalidad del Ciclo de Vida del Desarrollo de Software

2.3.1 Cómo ingresa un software a la organización

El software ingresa a una organización por cuatro caminos: i) adquisición; ii) integración o ensamblado; iii) desarrollo específico o iv) reingeniería de software existente.

Cada una de estas aproximaciones, a la concepción e implementación de software, tiene sus ventajas y desventajas, con fuerte impacto en los costos, mantenimiento y efectividad técnica. Y éstos suelen ser los factores preponderantes para decidir por una u otra opción. La seguridad del software no suele estar entre los factores relevantes a considerar. La seguridad es vista como un aspecto a considerar cuando la aplicación está siendo preparada recién para su despliegue y operación. Y es común aún que la seguridad del software no sea tenida en cuenta hasta que ocurre un incidente de seguridad que pueda asociarse directamente a una vulnerabilidad en el software.

El desarrollo de software específico desde cero, le da a una organización la posibilidad de reunirse con un producto que cumple exactamente con sus necesidades, a la vez que permite un control estricto durante las fases de operación y mantenimiento y sobre todos los aspectos del CVDS. Ya sea por un desarrollo propio o por contratos de tercerización. En este caso, para obtener una ventaja adicional de las herramientas ya utilizadas, muchas organizaciones han desarrollado extensiones que contemplan la seguridad en las metodologías existentes o procuran nuevas metodologías, enfocadas directamente sobre el aspecto seguridad puntualmente. Sin embargo hay poca evidencia empírica que muestre el retorno de la inversión o éxito de estas técnicas. Organizaciones como "*Carnegie Mellon Software Engineering Institute*" (SEI), "*Department of Homeland Security's*" (DHS) y el "*Department of Defense*" (DoD) están atentas a los métodos que puedan utilizarse a fin de estimar el "*Return on Investment*" (ROI) y ayudar a las organizaciones a decidir qué metodología "segura" podría ser aceptada para construir software a medida.

La reingeniería de software fue introducida por E.J. Chikofsky and A.H. Cross en 1990. Conceptualmente es la modificación de software existente para lograr que sea más efectivo y/o eficiente; cambiando, reemplazando o removiendo sus componentes. De acuerdo a los autores, la reingeniería puede involucrar ingeniería inversa. Los componentes o porciones de software necesitan de la ingeniería inversa para que el profesional pueda obtener un conocimiento abstracto y organizado del sistema existente. Luego se continúa el proceso de ingeniería directa, para reconstruir los mismos, o sumar nuevas funcionalidades. Esto nos enfrenta con muchos de los riesgos asociados a la adquisición de software, construido a medida y software ensamblado. Incluso las nuevas funcionalidades pueden ser provistas por COTS, ó hasta darse la ocasión en que algunas

modificaciones de código puedan integrar la nueva funcionalidad con partes de software que no ha sido modificado. Esta realidad potencialmente puede introducir nuevas vulnerabilidades en un sistema, debido a un conocimiento incompleto del diseño del software original. Por ejemplo, la actualización de una librería para proveer una extensión de sus funcionalidades puede afectar inadvertidamente su uso en otras partes del sistema ya que algunas partes del software pueden depender de la implementación original de la librería. Como conclusión, sin un conocimiento completo del sistema original, el software modificado puede resultar en salidas incorrectas con resultados inesperados y exposición a nuevas vulnerabilidades.

Si bien se realiza poca actividad de investigación sobre la seguridad en el ámbito específico de la reingeniería de software, tanto los profesionales que trabajan en la seguridad de software como aquellos que llevan adelante composición, adquisición y desarrollo a medida se benefician con los resultados de la ingeniería inversa. Gran parte de la investigación en esta área, se orienta al desarrollo de procesos estructurados de reingeniería, liderada por el SEI y la Universidad de Queensland, Australia.

Por otro lado, se debe tener en cuenta que una buena ingeniería de software es esencial para producir software seguro, pero no es suficiente. Esto es debido a que la mayoría de los esfuerzos persiguen metas orientadas a lograr funcionalidad, calidad o confiabilidad. Se puede argumentar que para el software, la seguridad es un aspecto esencial de la calidad y la confiabilidad, pero no siempre es considerado de este modo, incluso los procesos de software orientados a lograr calidad y confiabilidad omiten muchas de las actividades necesarias para lograr seguridad. Por ejemplo, el software puede ser cien por ciento confiable pero si asigna sus procesos de encriptación a un algoritmo como "*Data Encryption Standard*" (DES) o a uno propio, ese software debe ser considerado inseguro.

ISO/IEC 15288, CMM, y CMMI son ejemplos de recomendaciones para procesos de la ingeniería de sistemas que se enfocan en los procesos genéricos que una organización lleva adelante cuando desarrolla software. Estas metodologías proveen el marco de referencia mediante el cual se pueden definir procesos repetibles que potencialmente permiten ser medidos y mejorados para garantizar la calidad y costos de los procesos de desarrollo. Si la seguridad es considerada en el estudio de requerimientos, es posible utilizar estos modelos para mejorar la seguridad en sus productos de software mientras se mejora la calidad del software. De todos modos, muchas de las actividades requeridas para mejorar la seguridad de un sistema están separadas de aquellas realizadas rutinariamente para valorar la calidad de un sistema de software. Consecuentemente, muchas organizaciones proponen extensiones a estas recomendaciones a fin de considerar actividades referentes a la seguridad.

Es importante tener en cuenta que los desarrolladores de una aplicación pueden no estar elaborando las hipótesis adecuadas para desarrollar software seguro, debido simplemente a que la mayoría de las universidades no forman a sus egresados sobre la importancia de la seguridad, o sobre las actividades necesarias para desarrollar software seguro. Una iniciativa que intenta aportar una solución a este punto es el documento [Softare Assurance CBK 2007] o simplemente CBK. Intenta ser una "data" de entrada para que las universidades entrenen profesionales en esta área. Aún así, desarrolladores entrenados en seguridad pueden elaborar conceptos inapropiados. Por ejemplo, un concepto común es suponer que la seguridad será tratada con un sistema diferente o en una etapa de desarrollo diferente. Algunos conceptos inapropiados incluyen:

- Como el Sistema Operativo da soporte al control de acceso, el software no tiene que contemplar resolver aspectos en esta área y es seguro.
- Los host donde se ejecutará el software están en una red segura de modo que el software no necesita contemplar aspectos de seguridad.

- El software es un prototipo; la seguridad será enfocada en el desarrollo del producto final.
- El software encripta datos antes de transmitirlos, de modo que es adecuadamente seguro.
- El software está escrito en un lenguaje de tipos seguros (Por ej. Java o C#) de modo que es imposible que tengamos un problema de overflow, la seguridad ha sido considerada.

Para un subconjunto pequeño de aplicaciones, esto puede ser válido. De todos modos estas suposiciones pueden resultar no válidas al final del CVDS ya que el control de acceso mediante el SO puede estar deshabilitado; la red segura puede permitir el acceso de entidades no confiables; el prototipo puede impresionar muy favorablemente al cliente de modo que sea pasado directamente a producción; la encriptación puede ser muy fuerte pero el atacante podría estar interesado en atacar la interface de usuario o el código en java puede ser vulnerable a la inyección de comandos.

Esto nos lleva a concluir que es necesario revisar periódicamente el estado del software para determinar si las hipótesis originales referidas a la seguridad continúan siendo válidas. De no ser así, es recomendable tomar las decisiones para garantizar que dichas hipótesis se cumplan y garantizar que los cambios no han introducido nuevas vulnerabilidades.

2.3.2 Uso de métodos formales para obtener software seguro.

Los métodos formales son una adaptación a la construcción de software, de determinadas áreas de las matemáticas desarrolladas en el siglo XIX y XX. Un sistema formal se compone de cuatro elementos:

- Un conjunto de símbolos.
- Reglas para la construcción de formulas bien formadas en el lenguaje.
- Axiomas para formular postulados que sean verdaderos.
- Reglas de inferencia, expresadas en un metalenguaje. Cada regla de inferencia establece que una fórmula, llamada un consecuente, pueda ser inferida a partir de otra fórmula llamada premisa.

Los métodos formales, más que una metodología disciplinada, tratan de incorporar técnicas basadas en las matemáticas para la especificación, desarrollo y verificación de software. En la medida en que las vulnerabilidades puedan resultar de implementaciones funcionalmente incorrectas, los métodos formales, en general, mejoran la seguridad de software (a un costo). Un ejemplo exitoso y familiar de aplicación de métodos formales a la seguridad del software, es el chequeo de tipos, presente en los lenguajes modernos de programación como Java, C# y Ada5. Incrementa la tasa de detección de fallas y debilidades en tiempo de compilación y ejecución. Hay una “especificación” que contiene la información de tipo que provee el programador cuando declara las variables. Mediante el uso de algoritmos, tales como los propuestos por Robin Milner, se logra la “verificación” de la especificación, para inferir los tipos en cualquier parte del código y asegurar que el tipado sea internamente consistente. El resultado de la verificación es "la garantía" (que depende de la ausencia de intervenciones extrínseca) de la integridad en la interpretación de los bits crudos como valores abstractos dentro del software y de la integridad en las vías de acceso a esos valores. La verificación de tipos afecta a todas las prácticas de ingeniería de software que utilizan lenguajes modernos.

De todos modos, sigue siendo un desafío para los investigadores, desarrollar métodos formales para la especificación y verificación de propiedades de la seguridad no traceables sobre el software tales como la predecibilidad en el comportamiento del mismo frente a cambios inesperados en el medio ambiente donde se ejecuta, asociados con entradas maliciosas, inserción maliciosa de código o fallas intencionales.

Los métodos formales tienen en principio limitaciones de escala, entrenamiento y aplicabilidad. Para compensar las limitaciones de escala, han sido aplicados a partes seleccionadas o propiedades de un proyecto de software. Las limitaciones de entrenamiento se deben a que es difícil encontrar desarrolladores con la suficiente experiencia en lógica formal, rango apropiado de métodos formales para una aplicación o herramientas de desarrollo apropiadas para implementarlos. Finalmente, no todos los métodos formales son aplicables por igual a todos los sistemas. También tienen limitaciones de principios, en el sentido que una verificación formal puede probar que una descripción abstracta de una implementación satisface una especificación formal o que algunas propiedades formales están satisfechas en la implementación. De todos modos, un método formal no puede probar que una especificación formal captura el entendimiento intuitivo de los usuarios de un sistema y además no puede probar que una implementación corre adecuadamente en cada máquina física. Como ha mencionado Barry W. Boehm, [Boehm 81], los métodos formales son útiles para verificar un sistema, pero no pueden ser utilizados para validar un sistema. La validación muestra que un sistema estará de acuerdo a su misión operativa. La verificación muestra que cada paso en el desarrollo satisface los requerimientos impuestos en las etapas previas.

Los métodos formales han sido obligatorios para diseñar software que debe garantizar altos niveles de seguridad, por ejemplo, "*Common Criteria Evaluation Assurance Level 7*" (CC EAL 7) y superiores. También han mostrado ser exitosos en la especificación y testeo de sistemas pequeños muy bien estructurados tales como sistemas embebidos, algoritmos criptográficos, modelos de referencia de sistemas operativos y protocolos de seguridad.

2.3.3 Gestión de riesgos de la seguridad del software

Cuando se habla de gestión de riesgos del software, el término es utilizado habitualmente pensando en gestionar los riesgos de un proyecto o en gestionar los riesgos vinculados a la calidad del software. Así es utilizado por ejemplo en ISO/IEC 16085:2004. Y la mayoría de las metodologías y herramientas para gestionar proyectos y calidad del software no son fácilmente adaptables para gestionar aspectos referidos a riesgos que tengan que ver con la seguridad del software. Sólo recientemente, la gestión de riesgos de la seguridad del software a lo largo de todo su ciclo de vida, se ha convertido en un tema de amplia discusión, con técnicas y herramientas emergentes y la integración de la gestión de la seguridad del software con la gestión de riesgos de un proyecto de desarrollo de software. Así han surgido propuestas como la de [Viega 2001] quien sugiere impulsar las siguientes actividades como parte de los procesos de una gestión de riesgos de la seguridad del software a lo largo de su ciclo de vida:

- Derivación, elicitación y especificación de requerimientos de seguridad.
- Valoración de riesgos de seguridad.
- Diseño de una arquitectura segura.
- Implementación segura.
- Pruebas seguras.
- Garantizar la seguridad.

Otros autores [Morana 2006] sugieren un enfoque motorizado por actividades, a largo plazo y enfoques integrales sobre la seguridad del software, para mitigar el riesgo a corto plazo, mediante acciones sobre temas específicos de la seguridad. También recomienda que se estudie la seguridad del software en forma paralela con los riesgos en la seguridad de la información. Concretamente plantea un conjunto de actividades para gestionar el riesgo de la seguridad del software, vinculado directamente a las principales fases de su ciclo de vida:

- **Requerimientos**: Ingeniería de requerimientos de seguridad; establecimientos de metas a cumplir; aplicación de estándar de la industria; especificación de requerimientos técnicos de seguridad; modelado de amenazas y mediciones de la seguridad.
- **Arquitectura y Diseño**: Modelado de amenazas; patrones de seguridad de diseño y arquitectura; planificación de test de seguridad; revisión de la arquitectura y diseño de la seguridad y mediciones de la seguridad.
- **Desarrollo**: Revisión de código; utilización de patrones de seguridad; mitigación de errores y defectos; test unitario de seguridad; actualización de modelos de amenazas y mediciones de seguridad.
- **Testing**: Uso de patrones de ataque; utilización de pruebas automáticas de caja blanca y caja negra; test de regresión; test de stress; valoración de la seguridad de terceros; actualización de modelos de amenazas y mediciones de seguridad.
- **Despliegue**: Implementación de medidas operativas de seguridad; gestión de actualizaciones; actualización de modelos de amenazas y mediciones de seguridad.

Y a lo largo de todas las etapas del CVDS, recomienda tener en cuenta políticas de seguridad y llevar adelante entrenamientos específicos utilizando herramientas y métricas específicas.

Fernández [Fernandez 2006] describe una metodología de construcción de software seguro en donde la idea fundamental es aplicar principios de seguridad a cada etapa del desarrollo de software, y chequear en cada etapa el cumplimiento de esos principios, en coincidencia con la propuesta de [McGregor & Sykes 2001] que destaca el test en cada etapa de un desarrollo orientado a objetos. En la Figura 2.3.3.1 se puede ver un ciclo de vida de desarrollo de software seguro, donde las flechas blancas indican dónde debe ser aplicada la seguridad y las flechas negras indican dónde debe chequearse el cumplimiento de los principios de seguridad.

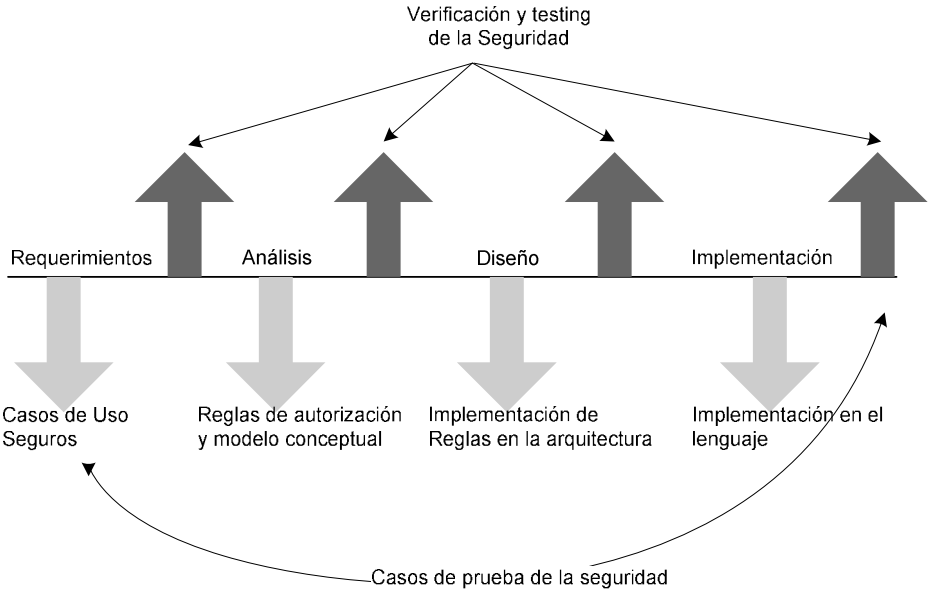

Figura 2.3.3.1. – Ciclo de vida de un desarrollo de software seguro.

De la etapa de requerimientos se obtienen casos de uso. De la etapa de Análisis se obtienen reglas de autorización que se aplican al modelo conceptual. En la etapa de Diseño se aplican las reglas

sobre la arquitectura. En la implementación, Despliegue y Mantenimiento se requieren lenguajes que permitan implementar las contramedidas identificadas. Se debe destacar que la verificación de la seguridad ocurre en cada etapa del desarrollo. En la Tabla 2.3.3.1 se muestra los métodos utilizados en cada etapa.

Etapa	Método para implementar la seguridad
Requerimientos	Casos de uso basados en la identificación de roles y análisis de ataques
Análisis	Modelo de patrones de la semántica de análisis
Diseño	Aplicación coordinada de patrones en las múltiples capas de la arquitectura
Implementación	Incorporar aplicaciones COTS seguras

Tabla 2.3.3.1. – Resumen de los métodos utilizados en cada etapa.

Las organizaciones han utilización durante años "*Risk Management Frameworks*" (RMF) y recientemente han incorporado la gestión de riesgos de seguridad y más recientemente aún se ha utilizado puntualmente para gestionar los riesgos de la seguridad en sistemas de software.

En el 2004, Microsoft describió su concepto de la gestión de riesgos de seguridad, incluyendo su propio "*Security RMF*" [Microsoft SRM 2006] el cual mapea actividades de la gestión de riesgos sobre fases del CVDS. Microsoft lo denomina "*MSF Process Model*", y se describe brevemente en la Tabla 2.3.3.2.

Fases del Proceso	Actividades del Framework de Gestión de Riesgos
Iniciación	**Iniciación de la definición del proyecto:** Todas las partes involucradas en la seguridad del proyecto deben definir objetivos, supuestos y restricciones. **Resultado:** Aprobación de la visión y alcances del proyecto.
Planificación	**Valoración y análisis del proceso de gestión de la seguridad:** incluye valoración organizacional, valuación de bienes, identificación de amenazas, valoración de vulnerabilidades y riesgos de seguridad y planificación de contramedidas. **Resultado:** Aprobación de la planificación del proyecto.
Construcción	**Desarrollo de las contramedidas de seguridad:** Desarrollo, test unitarios y valoración de la calidad de las contramedidas. **Resultado:** Finalización de la definición del alcance del proyecto.
Estabilización	**Pruebas de contramedidas de seguridad y funcionamiento de recursos:** Involucra la detección y valoración de la severidad de los bugs de seguridad. **Resultado:** Aprobar la liberación de una nueva versión del producto.
Despliegue	**Despliegue de políticas y contramedidas de seguridad:** Incluye la totalidad de la organización, ya sea que este centralizada o políticas específicas para sitios, y contramedidas para componentes de seguridad. **Resultado:** Finalización de la etapa de despliegue y reingreso en inicio del ciclo de vida, transfiriendo el conocimiento de la gestión de riesgos de seguridad y lecciones aprendidas.

Tabla 2.3.3.2. – Framework de Microsoft para la gestión de seguridad

La consultora Cigital (www.cigital.com) ha utilizado su propio RMF por más de una década y por contratación de DHS desarrolló una versión condensada [McGraw 2005], compuesta por cinco tareas básicas para orientar las actividades destinadas a mejorar la seguridad:

- Conocer el contexto del negocio.
- Identificar los riesgos técnicos y del negocio.
- Sintetizar y priorizar los riesgos, produciendo un conjunto ranqueado.
- Definir una estrategia para su mitigación.
- Llevar adelante las correcciones que sean necesarias y comprobar que son correctas.

Sumando una sexta actividad que impacta sobre estas cinco, que se refiere a la elaboración de mediciones y generación de informes.

Al igual que McGraw, Morana de Foundstone, una división de McAfee, propone un Framework para la Seguridad del Software, tal como el que muestra la Tabla 2.3.3.3.

Fases del CVDS	Requerimientos		Diseño	Desarrollo	Testing	Despliegue y Operación	
Mejores prácticas de la Seguridad del Software	Análisis preliminar de riesgos de Seguridad	Ingeniería de Requerimientos de Seguridad	Diseño conducido por los Riesgos de Seguridad	Implementación de códigos seguros	Test de la Seguridad	Despliegue y configuración de la Seguridad	Operación Segura
Actividades en curso del CVDS Seguro Aplicación de métricas y mediciones – Entrenamiento – Concientización							
Actividades del CVDS – S	Definición de casos de uso y mal uso	Definición de Requerimientos de Seguridad	Patrones de Arquitectura y Diseño seguros Planificación de Modelo de amenazas y Test de la Seguridad Revisión de la Arquitectura de Seguridad	Revisión de código por parte de pares Revisión automática de código estático y dinámico Test unitario de la Seguridad	Test funcional Test conducido por riesgos Test del sistema White Box Testing Black Box Testing	Configura-ción Segura Despliegue Seguro	
Otras Disciplinas	Valoración de riesgos de alto nivel		Valoración de riesgos técnicos				Gestión de incidentes Gestión de Cambios
Otras disciplinas en curso Gestión de riesgos de la información, cambios y vulnerabilidades							

Tabla 2.3.3.3. – Framework para Seguridad del Software

David Gilliam del "*Jet Propulsion Laboratory*" (JPL) de la NASA describe en [Gilliam 2004] una aproximación formal a la gestión y mitigación de riesgos de seguridad en el CVDS que requiere la integración de herramientas para la valoración de riesgos y herramientas para la gestión y mitigación

de riesgos, las que se usan de manera iterativa a lo largo del CVDS. Las herramientas que Gilliam describe son el "*Software Security Assessment Insrument*" (SSAI) desarrollado por el programa "*Reducing Software Security Risk*" (RSSR) de la NASA y la herramienta "*Defect Detection and Prevention*" (DDP) del JPL. Desde la aparición de esta iniciativa, hubo varias otras, privadas, que propusieron suites de herramientas para la gestión de riesgos de seguridad de aplicaciones, que funcionan de forma integrada y administradas centralmente, tales como "*APSolute Application Security*" de la firma "*Radware y Assessment Management Platform"* de SPI Dynamics, adquirida en el 2007 por HP y luego ofrecida como "*HP Application Security Center*". Estas soluciones incluyen herramientas que soportan capacidades como la revisión de código, valoración de vulnerabilidades, test de penetración, monitoreo y detección e ataques y intrusión, verificación y cumplimiento de objetivos de seguridad, gestión de políticas de seguridad, firewall, encriptación, valoración de vulnerabilidades y gestión de actualizaciones.

Como conclusión, todas ellas proponen considerar a la seguridad desde las etapas tempranas del CVDS: la elicitación de requerimientos. No existe propuesta de un framework que intente gestionar los riesgos de la seguridad, que no establezca esta regla como condición necesaria para desarrollar un software que garantice la seguridad. El punto aquí es ver cómo se resuelve la integración de la seguridad en dichas etapas, con qué elementos, cómo se resuelven los conflictos que puedan surgir, cómo son conducidas estas actividades hacia las etapas posteriores del CVDS. Nuevamente, en esta dirección este trabajo intenta proponer una alternativa novedosa.

2.3.4 Métricas de la seguridad

La seguridad del software y las aplicaciones de repente se han convertido en un gran negocio. Los defensores de los diferentes procesos para la generación de software seguro, las mejores prácticas para garantizar la seguridad del software, y una variedad de técnicas y herramientas de apoyo, sugieren que aquellos que las adopten obtendrán grandes beneficios en términos de mayor seguridad (o al menos menor vulnerabilidad) del software. Sin embargo, el hecho es que hay pocos indicadores concretos para medir objetivamente y con precisión la eficacia de estos diferentes procesos, prácticas, técnicas y herramientas. Por otra parte, hay en curso un debate animado sobre métricas de seguridad, en la comunidad que intenta definirlas, con respecto a exactamente qué puede y debe medirse como un indicador significativo de que el software es realmente seguro (o no vulnerable).

En la primera conferencia dedicada a las métricas de seguridad: Metricon 1.0, llevada a cabo en Vancouver, Columbia Británica, Canadá en agosto de 2006. Steve Bellovin, uno de los "barba gris" de la comunidad de especialistas en seguridad en internet, resumió muy bien el problema en su discurso de apertura de Metricon. Sostuvo que aún no es posible obtener métricas significativas de la seguridad del software:

> *Las cajas fuertes están clasificadas de acuerdo a cuánto tiempo van a resistir un ataque, bajo determinadas circunstancias. Podemos hacer lo mismo con el software...? Es bien sabido que cualquier pieza de software puede tener errores, incluyendo software que se asume que debería ser seguro. Esto significa que cualquiera sea la defensa, de un solo golpe bien colocado puede romperse. Podemos tener una defensa organizada en capas, pero una vez que una capa se rompe la capa siguiente queda expuesta, si es que ya no estaba expuesta la que tiene el problema. De este modo la resistencia de cada capa se aproxima a cero y adicionar más de ellas no ayuda. Necesitamos capas con una seguridad resistente, y no las tenemos. Así pues, muy a regañadientes mi conclusión es que, en el futuro inmediato, las métricas de seguridad*

son quimeras. Podemos desarrollar probabilidades de ocurrencia de vulnerabilidades, basándonos en cosas como el "Microsoft's Relative Attack Surface Quotient", el esfuerzo realizado en las auditorías de código, y cosas similares, pero no podemos medir con exactitud el esfuerzo necesario para quebrar la resistencia de un software.

De acuerdo al criterio de Bellovin, las métricas de seguridad del software, en términos de magnitudes absolutas, no son posibles, ya que no es posible tener certeza total de una magnitud absoluta asociada a la seguridad del software, ya que no se puede medir lo que no existe. Sobre la pista que dejó Bellovin, Jeremy Epstein de webMethods estuvo de acuerdo implícitamente con Bellovin en que la seguridad absoluta de software probablemente no es posible; pero no está de acuerdo en que es imposible hacer, en algún grado, una medición indirecta de la seguridad del software, recogiendo una combinación de estadísticas sobre el software (medidas que ya se han adoptado) y a continuación determinar cuál de estos parámetros (solo o en combinación con otros) dice en realidad algo significativo acerca de la seguridad del software.

Resumiendo: teniendo en cuenta una estadística como el número de fallas detectadas en el código fuente, es posible inferir "algo" acerca de su influencia sobre la seguridad del software compilado a partir de aquel código fuente. Cabría preguntarse: conseguir menos errores en el código fuente significa que el software es menos vulnerable...? Por cierto, es la premisa que da sustento a las herramientas de análisis de la industria.

Se puede argumentar, como lo hace Bellovin, que incluso una sola falla de implementación, si se explota, puede comprometer totalmente el software. Incluso si no hubiese fallas de implementación, el software puede mostrar debilidad general por deficiencias en su diseño y arquitectura. Insuficiencias mucho más difíciles de identificar y aún medir.

Los investigadores que trabajan en la definición de métricas de la seguridad del software no pretenden que puedan definirse medidas absolutas de seguridad. Lo mejor a lo que hoy se puede aspirar es a obtener resultados a partir de medidas indirectas. Medir diferentes características y propiedades del software que permitan inferir en conjunto un índice relativo de seguridad del software. De modo que se pueda comparar el mismo software operando en diferentes ambientes, o con otros productos de software similares operando en las mismas condiciones. En definitiva, medir es comparar.

Las métricas de la seguridad del software hoy en día, van en la línea sugerida por Epstein, es decir, recopilar estadísticas factibles de ser recopiladas, luego considerarlas y vincularlas a indicadores de la seguridad del software. Tienen que ver con investigar métricas de la seguridad de la información, calidad y confiabilidad para determinar si algunas de ellas pueden utilizarse para medir la seguridad del software. Esta es la aproximación del grupo de trabajo "*Software Assurance Program's Measurement*" del DHS el cual presenta un enfoque que intenta "aprovechar" las métricas de la arena de calidad del software (por ejemplo, CMMI) y del campo de la seguridad de la información (por ejemplo, CC, SSE-CMM, [121] NIST Special Publication (SP), 800-55, ISO / IEC 27004).

En marzo de 2003, Microsoft contrató los servicios del Departamento de Soluciones de Seguridad y Tecnología de Ernst & Young LLP para validar el modelo de Cociente Relativo de Superficie de Ataque "*Relative Attack Surface Quotient"* (RASQ) desarrollado por ellos mismos. Este indicador cuantifica la susceptibilidad relativa a los ataques que presenta cada una de sus plataformas de sistemas operativos. Desarrollado con la asistencia de "*Carnegie Mellon University*" (CMU) para compensar la falta de normas comunes para las métricas de software de seguridad, RASQ mide la vulnerabilidad de un sistema, en términos de la probabilidad de que un ataque contra el sistema efectivamente se produzca y sea exitoso. Para calcular el índice RASQ se deben identificar los

vectores de ataque, específicos de cada sistema, los que impactan positiva o negativamente su seguridad. Cada tipo de vector de ataque tiene un "peso" o factor de ponderación de ataque asociado, cuyo valor está entre 0 y 1. Indica el nivel de riesgo que corre el software, de ponerse en compromiso ese vector de ataque. El producto de ambos nos da una idea de la superficie de ataque vinculada a ese vector de ataque. Luego la sumatoria de todas ellas resulta en el índice RASQ. La puntuación final RASQ de un sistema es el producto de la suma de todas las superficies de ataque efectivas.

2.3.5 La Seguridad en diferentes modelos de procesos del CVDS

A lo largo de los años se han realizado muchos intentos por definir el modelo más eficaz para organizar la gestión, técnicas y actividades llevadas adelante a lo largo del CVDS. En la Tabla 2.3.5.1 se muestran los principales modelos, con ejemplos para cada uno de ellos.

Modelo	Ejemplos de implementación
Cascada (lineal secuencial)	Winston Royce [Winston 1970].
Iterativo e incremental	Joint Application Design (JAD) [Mei 1999].
Evolutivo	Ciclo de Vida Evolutivo, original de Tom Gilb (evolucionado a Evolutionary Systems Delivery, o Evo) [evo 2011], Rapid Application Development (RAD) [rad 2011], Genova [Arisholm 1999].
Espiral	Barry Boehm [Boehm 1984].
Versiones concurrentes	Modelo en cascada o de re-jerarquización [Sun 1996]
Proceso Unificado [Ambysoft-2006]	Rational Unified Process (RUP) [rup 2011], Agile Unified Process (AUP) [Ambysoft-2006_1], Enterprise Unified Process (EUP) [eup 2011].
Metodologías Ágiles	Ver punto 2.3.6.

Tabla 2.3.5.1. – Modelos para el CVDS

La mayoría de estos modelos, proponen una versión con mejoras, implementadas a través de "*frameworks*" o guías paso a paso, con actividades para todas las fases del CVDS a fin de promover desarrollos de software con la seguridad mejorada. Las metodologías que se describen aquí, o modifican actividades existentes del CVDS o suman nuevas actividades, con el objetivo de reducir el número de debilidades o vulnerabilidades e incrementar la seguridad del software frente a amenazas.

"*Comprehensive, Lightweight Application Security Process*" (CLASP) es una iniciativa de John Viega, de McAfee, Inc, que está diseñada para insertar metodologías de la seguridad en cada fase del CVDS. Tiene una licencia "*open source*" y su "*core*" es un conjunto de veinticuatro actividades enfocadas en la seguridad que pueden ser integradas dentro de cualquier proceso de desarrollo de software. Entre las veinticuatro actividades se incluyen:

- Monitoreo de métricas de seguridad.
- Identificación de requerimientos y roles de los usuarios.
- Investigación y valoración de soluciones a la seguridad.
- Análisis de seguridad del diseño del software.
- Identificación e implementación de pruebas de seguridad.

La actividad inicial propuesta por CLAP pertenece a la gestión del proyecto. La introducción de un programa de concientización sobre la seguridad pretende ser algo más que la simple formación de los desarrolladores que se ocupan directamente de los requerimientos de seguridad. Todos los "*stakeholders*" que están expuestos directamente al proceso de desarrollo, deben recibir una formación básica sobre la seguridad, que les permita tomar conciencia de la exposición a la que puede quedar sometida la organización si no se consideran los aspectos de seguridad. Concretamente deben comprender los costos directos asociados a las actividades relacionadas a la seguridad así como los beneficios a mediano y largo plazo de una postura orientada a mejorar la seguridad. De lo contrario,

cuando el proyecto comienza a deslizarse sobre sus etapas, las actividades relacionadas a la seguridad estarán entre las primeras a ser diferidas si no se ve claro el impacto concreto que ellas tienen sobre el producto.

CLAP menciona que la tarea principal, para alguien que especifique los requerimientos de seguridad, es identificar a un alto nivel, el modelo se seguridad básico para el producto de software. Por ejemplo, el profesional debería comenzar identificando los recursos que podrían estar en riesgo, las funciones y responsabilidades de los usuarios que acceden a esos recursos y las posibles consecuencias para la organización, si estos recursos llegan a estar comprometidos. Estas actividades no sólo proporcionarán un contexto para la toma de decisiones, que impactarán sobre todo el CVDS, sino que también ayudarán a definir un marco para la rendición de cuentas, sobre un proyecto que contemple objetivos de seguridad que se encuentren dentro del diseño del producto.

La mayoría de las actividades de la seguridad, tradicionalmente asignadas a los implementadores hoy en día son mejor manejadas por los arquitectos y diseñadores de software. De modo que la mayoría de los aspectos de la seguridad podrían ser asignados a tiempos de arquitectura y diseño, lo cual resulta más rentable. Esto también permite concentrar el manejo de la seguridad en un conjunto reducido de miembros confiables de la organización.

Varias tareas claves son asignadas a un auditor de la seguridad, el cual es un rol nuevo que introduce CLASP en CVDS. La creación de este nuevo rol se justifica en el hecho de que muchos equipos de desarrollo pueden fácilmente vincularse estrechamente con sus desarrollos como para proceder a analizarlos directamente. Incluso de acepta como una buena práctica incluir auditores independientes de terceras partes. Estas valoraciones también están entre las actividades más simples y más rentables que una organización puede introducir para mejorar su postura frente a la seguridad y reforzar los esfuerzos de sus equipos internos, simplemente designado en estas tareas, miembros de la misma organización asignados a otro producto.

CLASP tiene un fuerte impacto en varias de las actividades tradicionales del CVDS, tales como la especificación de requerimientos. Materialmente no cambia los pasos a dar en estas etapas. En su lugar recomienda extensiones a los artefactos más comunes y ofrece guías de implementación para contenidos específicos de la seguridad. Para mayor información sobre CLASP se puede consultar [CLASP 2005].

"*TSP Secure*" es otra iniciativa conjunta entre el "*Process Program*" del SEI y el CERT. El objetivo es extender el proceso TSP para desarrollar sistemas de software seguro. Esta iniciativa considera que un sistema es seguro cuando está libre de vulnerabilidades conocidas. Estas vulnerabilidades resultan de errores en la especificación de requerimientos, diseño e implementación.

La iniciativa "*Secure Coding Initiative*" (SCI) del CERT conlleva un alto grado de colaboración y esfuerzo basado en la comunidad, a fin de desarrollar estándares de codificación segura y herramientas, para lenguajes de programación de uso general como C, C++, Java, así como el apoyo a la adopción de esas normas. "*C Secure Coding Standard*" del CERT [Seacord 2008] establece una guía para la codificación segura en C.

"*Team Software Process*" (TSP) ofrece una aproximación a un desarrollo disciplinado que miles de equipos ya han utilizado para producir software casi sin defectos. Incluye un amigable framework para planificar, medir y gestionar la calidad. Este "*framework*" soporta y habilita para el desarrollo e implementación de procesos y estándares de desarrollo de software. Soporta el uso de procesos y estándares de diferentes formas, planificación para lograr calidad, para seguimiento y gestión del plan de desarrollo, para grupos de desarrollo auto-dirigidos con objetivos comunes y gestión y mitigación de riesgos. "*TSP Secure*" es una especialización de TSP con una especial atención en la seguridad del software.

El proceso de desarrollo con "TSP Secure" requiere tanto entrenamiento en TSP como en prácticas de desarrollo seguro. El entrenamiento en TSP incluye a personas puntualmente y al equipo

en su conjunto. El entrenamiento apropiado en prácticas de desarrollo seguro, en contraste, es fuertemente dependiente de lenguajes, librerías y sistemas operativos a utilizar como también de los ambientes de desarrollo y ejecución. Hoy existe material para poder entrenar profesionales en la codificación segura con C y C++.

Dado que el objetivo de "TSP-Secure" es vasto, el foco inicial está sobre las prácticas de codificación segura y en la incorporación de herramientas de análisis estadístico en los procesos de desarrollo.

2.3.6 Metodologías ágiles y desarrollo de software seguro

Los métodos ágiles están caracterizados por lograr la satisfacción del cliente mediante entregas tempranas y frecuentes de software útil y viable (en pocas iteraciones); un desarrollo iterativo; aceptación de cambios tardíos en los requerimientos; integración de personal del cliente y de negocios en el ambiente de desarrollo; desarrollo en paralelo de múltiples versiones y equipos auto-organizados.

De acuerdo con Konstantin Beznosov, [Beznosov 2006] los métodos ágiles son iterativos por naturaleza. No siguen el método lineal tradicional de desarrollo de requerimientos, diseño, implementación y fases de testing. En su lugar los métodos ágiles repiten la secuencia "tradicional" varias veces. Desde este punto de vista, pueden ser comparables con el modelo en espiral. Los métodos ágiles también destacan una aproximación evolutiva hacia la producción de software (construir algo pequeño, testear algo pequeño, tener un campo de trabajo pequeño) con varias actividades del ciclo de vida ocurriendo de forma concurrente.

Como está caracterizado por el "Manifiesto Ágil", todos los métodos ágiles tienen un objetivo primordial simple: producir software funcionalmente correcto, tan rápido como sea posible. Por esta razón, las metodologías ágiles evitan actividades del ciclo de vida que:

- No estén involucradas directamente con la producción de software (por ejemplo, la generación artefactos como documentación, la cual es necesaria para la mayoría de las actividades de evaluación y validación).
- No puedan ser realizadas por miembros del equipo de desarrollo y de forma concurrente con otras actividades del ciclo de vida.
- Requieran un conocimiento experto más allá del esperado en un equipo de desarrolladores. Metodologías ágiles no contemplan la inclusión, dentro del equipo de software, de un experto en seguridad u otro personal especializado que no sea desarrollador.
- Hagan foco en cualquier actividad que no sea la producción de software correcto rápidamente. Los proyectos ágiles tienen dificultad para incorporar objetivos no funcionales tales como confiabilidad y seguridad.
- Deban ser realizadas en un ambiente con restricciones sobre quién debe trabajar en el proyecto o sobre un role o condiciones de trabajo particulares. Los proyectos ágiles no incluyen ni se acomodan fácilmente a conceptos tales como separación de roles y responsabilidades, menor privilegio y control de acceso basado en roles sobre los artefactos de desarrollo.

Se ha debatido in extenso sobre si es posible producir software seguro para los proyectos de software que utilizan métodos ágiles. En el Anexo F de [SOAR 2007] se pueden ver algunos ítems frecuentemente citados en relación con los desarrollos ágiles así como los contraargumentos sobre cómo los métodos ágiles pueden beneficiar la seguridad del software. También se describen algunos

esfuerzos para definir métodos ágiles adaptados que brindan un mayor soporte al desarrollo de la seguridad.

2.4 Requerimientos de seguridad del software

La fuente de los problemas que impactan en los requerimientos de seguridad, no está limitada exclusivamente al dominio de la seguridad. Si bien muchos de los problemas se originan en una inadecuada e imprecisa especificación de requerimientos, también es real que una inadecuada interpretación de los mismos, en etapas posteriores del CVDS: concretamente en la implementación, acarrea sus consecuencias. Hacer ingeniería de requerimientos es duro y hacerla bien es más duro aún. Incluso para pequeños proyectos de software, puede ser necesario definir una gran cantidad de requerimientos, lo que se traduce en una inversión significativa para la totalidad del proyecto. Y este número crece exponencialmente con el tamaño del proyecto, su complejidad y su proceso de integración en ambientes de producción.

Entre los aspectos específicos de la seguridad que impactan en la ingeniería de requerimientos, se incluyen:

- Las personas involucradas con el proyecto no están informadas o no les interesa puntualmente el aspecto requerimientos de seguridad. La tendencia es a asumir que las necesidades referidas a la seguridad están garantizadas (por quién..?).
- Las guías y técnicas tradicionales tienden a enfocarse exclusivamente sobre los requerimientos funcionales.
- Las acciones tendientes a garantizar la seguridad del software son percibidas como limitaciones a la funcionalidad o interferencias con la usabilidad.
- Resulta más difícil especificar qué no debe hacer un sistema, que aquello que debe hacer de manera segura.
- Los "*stakeholders*" deben procurar entender las amenazas a las que se enfrenta el software para construir las defensas apropiadas. Por otro lado, las amenazas a las que se enfrenta el producto en el entorno de producción, son muy diferentes a las que se encuentra en el ambiente de elicitación de requerimientos. Las amenazas evolucionan.
- El perfil de las personas, que colaboran para definir el sistema, no es típicamente el perfil de las personas que intentarán abusar del sistema, y que son precisamente a quien deben enfrentar las protecciones que se construyan. Para ello se requiere creatividad, experiencia y una mentalidad diferente (muchas veces vinculada a lo lúdico) que sería interesante poder disponer al momento de definir los eventos que podrían comprometer la seguridad del software.
- Pueden existir problemas relacionados a la seguridad que impacten directamente al equipo de desarrollo: por ejemplo el acceso a información de servicios contratados, o a los cuales deba integrarse el software. Partes del desarrollo puede estar tercerizado y la información referida a evaluación de riesgos y vulnerabilidades ser de acceso restringido o estar clasificada. Esto impide el acceso a la información de los desarrolladores y por lo tanto a un conocimiento pleno de la misma.
- Ausencia de conocimiento sobre cómo impacta un componente en un gran sistema al cual se integrará. El componente se comporta de manera segura cuando opera aislado en su entorno de desarrollo, pero no así cuando se lo ensambla con el resto en el entorno de producción.

- Dudas respecto a quién tendrá que rendir cuentas (profesional y legalmente) en el supuesto caso de que el software sea inseguro. Actualmente los desarrolladores tienen responsabilidad (al menos algunos) sobre la funcionalidad del software que implementan. Puede extenderse esta política sobre aquellos aspectos que hacen a la seguridad ? Y quién se hace responsable sobre aquellos requerimientos de seguridad no especificados en primera instancia, aún cuando se conocen los riesgos del ambiente en donde trabajará?

Hacer ingeniería de requerimientos de software seguro, si bien es recomendable ampliamente, es obligatorio en las siguientes disciplinas:

- Software vinculado a la seguridad de las personas, como pueden ser dispositivos médicos y aquellos utilizados en usos pacíficos de la energía nuclear.
- Software que deba ser tolerante a fallas o del cual dependa la supervivencia de las personas, como pueden ser sistemas telefónicos o de comunicaciones de emergencia o para situaciones límites.
- Sistemas embebidos, con aplicación a la industria espacial y de armamentos.

Las técnicas y soluciones aportadas por la ingeniería de requerimientos han mostrado no ser suficientes para garantizar que un software sea seguro ya que tienden a contemplar contra medidas para prevenir peligros naturales que son de carácter estocástico. No tienen en cuenta que el software debe poder defenderse contra usuarios malintencionados y ataques dirigidos, totalmente intencionales: amenazas. La arena de las amenazas es un dominio más dinámico y menos predecible que el simple azar.

Hay que tener en cuenta, que conceptualmente la seguridad, ha sido incluida con otros requerimientos no funcionales (RNF) de software, como rendimiento y calidad. Durante la etapa de requerimientos, los RNF son capturados y definidos como atributos del software pero el proceso termina ahí. Los RNF necesitan ser mapeados como requerimientos funcionales de modo que permitan ser construidos en el software y testeados apropiadamente. Por ejemplo, un requerimiento como: "*El software no debe ser susceptible a un buffer overflow*" debería ser mapeado sobre requerimientos funcionales para validar entradas y utilizar lenguajes con tipado seguro y uso seguro de memoria. Algunas adaptaciones de casos de uso a casos de abuso y mal uso hacen este paso de mapeo explícitamente.

Mapeando, relacionando requerimientos no funcionales con requerimientos funcionales, la seguridad pasa a ser parte del proceso de análisis de requerimientos en general. Los potenciales conflictos de intereses, diseño, implementación, entre ambos pueden ser identificados y resueltos. Esto no significa que necesariamente todos los RNF deban ser mapeados como funciones del software. Pueden existir otros que deban solucionarse a nivel de abstracción de la aplicación y no del software puntualmente.

Cuando se habla de requerimientos para lograr objetivos de seguridad a menudo se confunden con los requerimientos de que un software sea confiable. Los primeros incluyen funciones que implementan una política de seguridad. Estas son las áreas funcionales que corresponden por ejemplo al control de acceso, identificación, autenticación y autorización, a funciones de encriptación/desencriptación y gestión de claves. Funciones para impedir la violación de las propiedades de seguridad de un sistema o de la información, tales como el acceso no autorizado, modificación, denegación de servicio, divulgación, etc. También se habla en términos de requerimientos de servicios de seguridad. Los segundos afectan directamente la probabilidad de que un software en sí sea confiable, lo cual está ligado a requerimientos no funcionales que en su conjunto

garanticen que el software seguirá siendo previsible aún cuando su previsibilidad se vea amenazada por razones estocásticas, no intencionales, no patrocinadas. Estos requerimientos a menudo apuntan a reducir o eliminar vulnerabilidades ligadas al proceso, al plan de desarrollo y a la gestión del proyecto. Estos requerimientos tienen que ver con acciones tales como la validación de entradas, el manejo de excepciones, etc. Ambas categorías se ven significativamente influenciadas por el conocimiento del ambiente de amenazas, el cálculo de riesgos y las estrategias de identificación y mitigación de los mismos.

Por otro lado, la fase de requerimientos cubre todas aquellas etapas que deben ocurrir necesariamente antes de la etapa de diseño. Esto incluye la especificación del sistema, con las funciones asignadas al software y los planes para el proyecto y el desarrollo. Las entradas pueden ser de diversas fuentes, incluyendo a los clientes, usuarios, administradores, grupos de aseguramiento de la calidad y testing, y una visión de cómo serán tratados los requerimientos del software. El principal producto de la fase de requerimientos es una especificación de requerimientos que defina los aspectos funcionales y no funcionales del producto software. El final de la etapa está marcado por la aceptación de la especificación por parte del cliente y otros "*stakeholders*". Por otro lado, en un ambiente donde las condiciones del negocio son siempre cambiantes, los requerimientos pueden ser modificados en el transcurso de la construcción del software y es necesario un esfuerzo para mínimamente identificar estos cambios.

En la Figura 2.4.1 se muestra una vista general de las tareas y artefactos involucrados en la fase de requerimientos de un desarrollo de software.

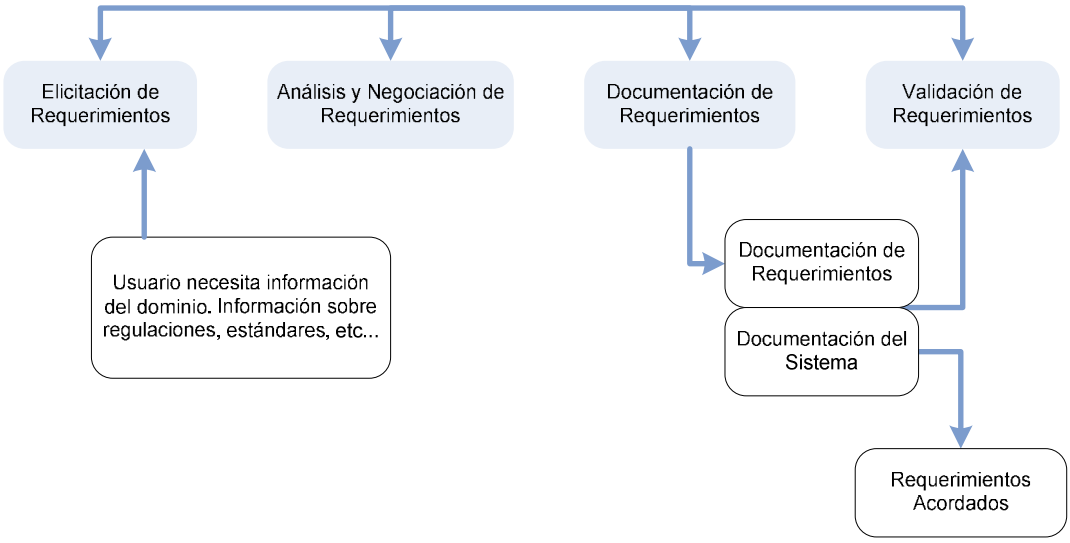

Figura 2.4.1. – Tareas y artefactos de la ingeniería de requerimientos

Existe un rico cuerpo de trabajo y resultados en la Ingeniería de Requerimientos, así como herramientas y técnicas para dar soporte a los procesos, pero desafortunadamente la mayoría de los trabajos no tienen en cuenta, explícitamente, la seguridad. Y los trabajos que habitualmente consideran aspectos de requerimientos de seguridad, lo hacen en términos de funcionalidades requeridas por el sistema como puede ser un control de acceso. La ingeniería de requerimientos de seguridad, es una propiedad emergente de los sistemas de software y si bien la implementación de las funcionalidades de la seguridad requerida por un sistema puede satisfacer muchos de los requerimientos de seguridad en

términos de una propiedad del software, será necesario un análisis diferente para atender estos diferentes objetivos.

El propósito es destacar las diferencias entre las actividades de la fase de requerimientos como habitualmente es realizada y cómo puede ser adaptada y aumentada para incrementar la seguridad de un producto de software a desarrollar. En la Figura 2.4.2 se muestran ejemplos de actividades y artefactos adicionales para incrementar la seguridad del software, sobrepuestos a los procesos genéricos de una etapa de requerimientos.

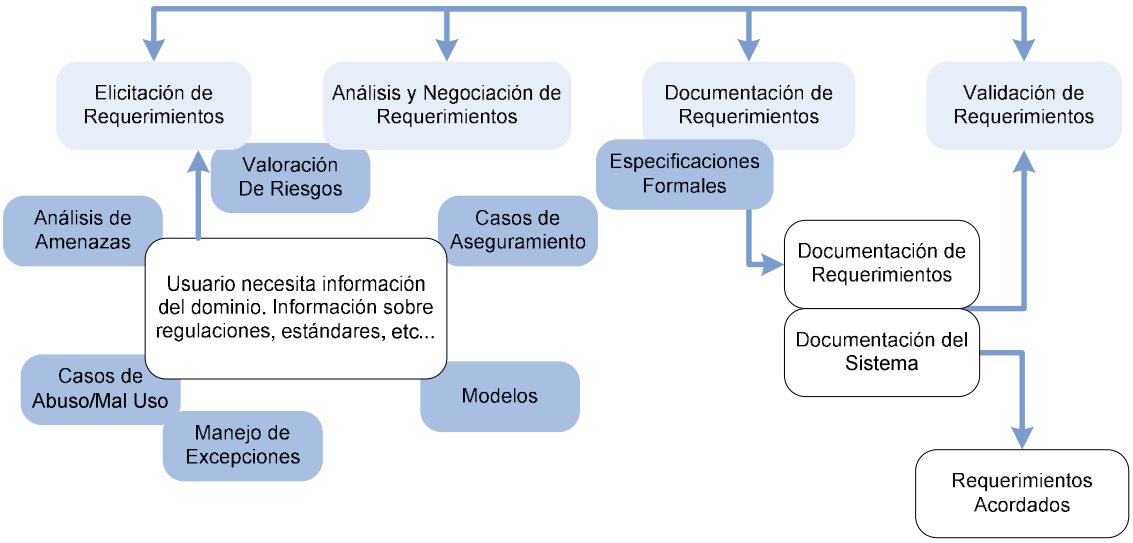

Figura 2.4.2. – Procesos de la IR (Adicionales para la construcción de software seguro)

La fase de requerimientos produce dos tipos de documentos:

1. La especificación, que es el producto final del trabajo.
2. La justificación que documenta el proceso y las decisiones que llevaron a la especificación. Esta justificación es utilizada para construir el producto seguro y también es útil para las etapas de desarrollo cuando se intentan introducir cambios.

Por otro lado, los requerimientos son un factor importante a lo largo de todo el CVDS. En la Tabla 2.4.1 se muestran las interpretaciones de las actividades y aspectos relacionados a los requerimientos de seguridad, hecha en las etapas posteriores a la fase de requerimientos.

Aspectos de los Requerimientos	Interpretaciones para la Seguridad
Trazabilidad, por ejemplo, garantiza que todos los elementos del diseño y el código son derivados de algún requerimiento.	Previene que el software entregado tenga funciones no especificadas.
Verificación del diseño y el código, para garantizar que se implementan todos los requerimientos.	Asegura que el software entregado tenga todas las características y propiedades de seguridad que fueron especificadas.
Control y gestión de cambios.	Permite analizar cómo afecta la seguridad los cambios propuestos, directamente o por efecto dominó.
Actualización de la documentación para reflejar los cambios.	Mantiene actualizado el análisis de amenazas para reflejar los cambios en el ambiente de amenazas.

Tabla 2.4.1. – Requerimientos a lo largo del CVDS

2.4.1 Expertos en seguridad en el grupo de estudio de requerimientos

Un aspecto que se observa cada día más demandante, es la incorporación de expertos en seguridad en los equipos de desarrollo de software. Si bien no todos los roles mencionados abajo tendrán el mismo nivel de involucramiento, todos deberían tener acceso a revisar los resultados de la etapa de especificación de requerimientos. A diferencia del equipo convencional para la etapa de requerimientos, formado por "*stakeholders*", usuarios, desarrolladores, líder de proyecto, arquitecto de software, personal encargado de garantizar la calidad y testeadores, estos roles adicionales cumplen una función de consultores y/o consejeros, necesitando en ocasiones de su visto bueno para poder continuar con la implementación de un diseño de software.

- **Ingenieros en Seguridad:** Pueden ser útiles al momento de tener que decidir aspectos de seguridad que se incorporarán al software y otros que serán resueltos a nivel de aplicación o sistema.
- **Analistas de riesgo:** Junto a la información de análisis de amenazas, son elementos claves al momento de tomar decisiones sobres aquellos aspectos que afectan la priorización de requerimientos de la seguridad del software.
- **Certificadores y Auditores:** Esencial para aquellos productos de software que deban ser certificados o acreditados en sus requerimientos ya sea que deban cumplir con un estándar o un nivel de aseguramiento dado.
- **Expertos en aseguramiento de la información:** Expertos en estas áreas con experiencias en la defensa y reparación de software existente, pueden ofrecer profundidad en cómo definir y construir software nuevo.

2.4.2 Buenos requerimientos y buenos requerimientos de seguridad

En la Tabla 2.4.2 se muestra un conjunto comúnmente aceptado de características que poseen los buenos requerimientos en general, junto con aquellas extensiones aplicables a, o esperadas de, los requerimientos de seguridad de software.

	Concepto convencional de bondad de los requerimientos	Concepto de la comunidad de aseguramiento del software
Corrección Hace lo que se le asignó?	– La funcionalidad a ser entregada está definida con precisión. – No hay conflictos con otros requerimientos.	– El requerimiento describe cómo se comporta cuando falla (manejo de excepciones). – Pueden identificarse restricciones y otros requerimientos.
Viabilidad Puede hacerlo?	– Cada requerimiento puede ser implementado dentro de las capacidades y limitaciones conocidas para el sistema y su medio ambiente.	– Las amenazas están incluidas en la lista de limitaciones conocidas.
Necesidad Es necesario?	– El requerimiento documenta algo que el cliente realmente necesita o es requerido para dar una conformidad.	– La funcionalidad es requerida para asegurar el software.
Sostenible Por qué es importante el requerimiento?	– Razón que justifique la necesidad y/o propósito para que el requerimiento sea documentado.	– La justificación es importante para construir el caso de aseguramiento. – Documentar la razón de la seguridad ayuda a garantizar que al requerimiento no se le asigne una baja prioridad al momento de ser implementado.

Prioridad Es más o menos importante que otros?	– Qué tan esencial es cada requerimiento, característica o caso de uso para definir una versión específica del producto. – Valoración típica de priorización: ALTA = Entra en la próxima versión del producto; MEDIA = Se difiere para próximas versiones; BAJA = Deseable pero no necesario.	– El análisis de amenazas y valoración de riesgos afectará su priorización.
Preciso Se interpreta de manera única?	– Sólo puede extraerse una interpretación.	
Verificable Puede ser testeado, traceado y medible?	– Capacidad para testear que cada requerimiento está implementado apropiadamente. – Capacidad de trazar un requerimiento entre las fase en que es implementado. – Capacidad de medir/mostrar/poner en evidencia de que cada requerimiento acordado ha sido implementado.	– Se necesitan test para demostrar que ciertos comportamientos no han sido implementados, que x no ocurre en presencia de y. – Se necesitan test para demostrar que las restricciones también se cumplen.

Tabla 2.4.2. – Características de una efectiva declaración de requerimientos

Los requerimientos de software en general son requerimientos que apuntan a lograr una funcionalidad deseada, y en algunos casos, pueden ser requerimientos para lograr una funcionalidad de rendimiento esperada (Ej.: "la función debe ser completada en *n* microsegundos"). Ellos tienden a ser expresados en términos positivos: "el software debe…".

En contraste, los requerimientos de seguridad, particularmente seguridad del software, tienden a ser restricciones, a una funcionalidad o a una expresión sobre la necesidad de una propiedad o atributo, que debe manifestar el comportamiento del software. Al menos inicialmente en el CVDS, durante la captura de requerimientos, ellos se manifiestan en términos negativos.

Como con todo software, el proceso de capturar requerimientos para la seguridad requiere de múltiples iteraciones de elicitación y análisis. Los Ingenieros en Requerimientos no deben limitarse por convenciones que recomienden no incluir sentencias en negativo o requerimientos que a primera impresión puedan parecer inalcanzables. Estas necesidades deben ser capturadas, analizadas y convertidas en requerimientos funcionales: acciones concretas, positivas. Por ejemplo el requerimiento "*el software no debe ser susceptible a un buffer overflow*" está en términos negativos y resulta a primera impresión inalcanzable, pero es necesario que se cumpla para que el software sea seguro. Será necesario expresarlo como requerimiento para lograr la funcionalidad que previene un buffer overflow, por ejemplo: validar entradas, gestionar de forma correcta la memoria, manejar las excepciones, etc.

2.4.3 ¿Dónde y cómo se originan los requerimientos de seguridad?

El documento Software Assurance (CBK) del DHS describe varias categorías de necesidades a ser tenidas en cuenta al momento de recolectar requerimientos de seguridad:

- Necesidades vinculadas a la seguridad de los "*stakeholders*"
  - Protección de sus bienes.
  - Necesidades surgidas a partir de amenazas al medio ambiente donde funcionará el software.
  - Necesidades de usabilidad, confiabilidad y disponibilidad.
- Necesidades vinculadas a la resistencia y supervivencia frente a ataques

- Necesidades de sostenibilidad del software.
- Necesidades de validación, verificación y evaluación.
- Necesidades de desalentar ataques.
- Necesidades de certificación, acreditación y auditoría.

Una necesidad no es un requerimiento. Una vez identificada, la necesidad debe ser analizada, priorizada y especificada antes de que se convierta en un requerimiento. En la tabla 2.4.3 se muestra como se utilizan los métodos convencionales de elicitación y análisis de requerimientos para dar soporte a la especificación de requerimientos de seguridad.

Método Convencional	Ejemplo de Extensión para la Seguridad
CONOPS	Trabajo desarrollados en CONOPS para la seguridad [Ammala 2000].
Despliegue de Funciones de Seguridad	Sin relación.
Descomposición Funcional	Identificación de amenazas a lo largo del flujo de datos.
Descomposición Orientada a Objetos	Métodos orientados a aspectos.
Desarrollo de Casos de Uso	Casos de Mal Uso y de Abuso.
Estudio de negocio	Identificación de amenazas y ataques únicos para el dominio de aplicación (tales como aplicaciones web).
Simulaciones	Adquisición basada en simulación, su principal impulsor es el DoD.
Modelados	Modelo de amenazas, también modelo orientado a aspectos.
Prototipado	Métodos orientado a los datos.
Sin relación	Árbol de ataques/amenazas.

Tabla 2.4.3. – Extensión a Métodos Convencional de Requerimientos

En el año 2003, A.Rashid [Rashid 2003] propuso el uso del desarrollo de software orientado a aspectos para mapear uno o más requerimientos funcionales sobre cada requerimiento no funcional que afecta al requerimiento funcional, con miras a considerar la seguridad directamente como uno de los varios tipos de requerimientos no funcionales.

Axel van Lamsweerde [Lamsweerde 2003] propone el método KAOS para modelar los requerimientos de seguridad y protección mediante el uso de anti-objetivos. Un anti-objetivo describe las vulnerabilidades que hacen imposible cumplir las necesidades establecidas para el sistema. Los requerimientos de seguridad resultantes son expresados en términos de "evitar" anti-objetivos a fin de eliminar vulnerabilidades que pondrían en riesgo el alcance de los objetivos del software.

Ian Alexander [Alexander 2002] propone también evitar vulnerabilidades, pero confía en casos de mal uso en lugar de los anti-objetivos. John McDermott [McDermott 2001] sustituye casos de mal uso por casos de abuso. Otros autores como H.In y Barry Boehm hacen adaptaciones de la estrategia ganar-ganar aplicada a la gestión de la calidad para incluir los requerimientos de seguridad.

Charles Haley [Haley 2004] propone representar los requerimientos de seguridad como una descripción transversal de amenazas, las que a su vez ayudan a componer estos requerimientos con los requerimientos funcionales del sistema; la especificación resultante define un conjunto de restricciones sobre los requerimientos funcionales. Estas restricciones son, para el autor, los requerimientos de seguridad. Este es un trabajo que pretende dar una respuesta a los problemas que surgen con otras

aproximaciones al integrar los requerimientos de seguridad con otros tipos de requerimientos. Haley menciona los siguientes problemas:

- No hay una definición simple para "requerimientos de seguridad". En algunos casos el término se refiere a requerimientos para lograr funcionalidades de seguridad. En otros casos se refiere a restricciones en la funcionalidad. Y en otros casos, se relaciona a la necesidad de que la funcionalidad del software opere de manera consistente con políticas de seguridad que gobiernan la aplicación. Más aún, los requerimientos de seguridad se ven como "anti-requerimientos" o "anti-patrones", definidos en términos de las vulnerabilidades que deben evitar para que la aplicación pueda satisfacer requerimientos de la no-seguridad. Esta sobre abundancia de definiciones sobre los requerimientos de seguridad hace difícil de determinar qué aproximación de la ingeniería de requerimientos está mejor posicionada para este problema en particular.
- Cuando se definen requerimientos de seguridad, son inconsistentes y los criterios para determinar si han sido satisfechos, son difíciles de comprender.
- No hay un camino claro aún para derivar los requerimientos de seguridad a partir de los objetivos o necesidades del negocio.

John Wilander y Jens Gustavsson [Wilander 2005] están de acuerdo con Haley en que hay una práctica muy pobre en relación a los requerimientos de seguridad. Ellos observan adicionalmente que la seguridad es tratada como un aspecto funcional compuesto de características de seguridad tales como login, backup, y control de acceso. Los requerimientos para garantizar la seguridad del software no son tenidos en cuenta. Luego los autores elaboran la siguiente conclusión a partir del estudio de la mayoría de los requerimientos de seguridad:

- Los requerimientos de seguridad están ausentes o pobremente especificados debido a un inadecuado entendimiento de la seguridad en general y de la seguridad como una propiedad particular, por parte de la mayoría de los ingenieros que hacen requerimientos.
- La selección de los requerimientos de seguridad es inconsistente con la mayoría de los requerimientos funcionales. A partir de tener en cuenta los requerimientos de seguridad se suman inconsistencias debido a que mientras que algunos requerimientos son incluidos, otros de los cuales dependen, son dejados de lado.
- El nivel de detalle observado de las sentencias que definen los requerimientos es inconsistente.
- La no observación de políticas de seguridad resulta en soluciones personalizadas que presentan limitaciones en su efectividad al prevenir vulnerabilidades por un inadecuado conocimiento de la seguridad por parte de los desarrolladores.

En un trabajo posterior [Haley 2006] Haley describe un framework para atacar los requerimientos de seguridad, combinando lo que él identifica como una aproximación entre la ingeniería de requerimientos y una aproximación de la ingeniería de seguridad. Los objetivos de seguridad definidos expresan la necesidad de proteger la aplicación de daños y amenazas. Estos objetivos son materializados en los requerimientos de seguridad, los que son en esencia, restricciones a los requerimientos funcionales, cuyo resultado es un adecuado nivel de protección. Finalmente, el framework especifica la necesidad de desarrollar "argumentos de satisfacción" que demuestren la habilidad del sistema para funcionar consistentemente con los requerimientos de seguridad.

2.4.4 Métodos, técnicas y herramientas para la Ingeniería de Requerimientos de la Seguridad del Software

Este título describe algunos métodos utilizados en proyectos de desarrollo de software en la industria, o en "pruebas piloto" exitosas en el área de investigación, y que han sido considerados como una tecnología lista para ser transferida. Estas técnicas hacen posible considerar la seguridad en los requerimientos de software, con herramientas que implementan o dan soporte a la técnica. Mirando críticamente, todas ellas hacen uso de la misma información. Realizan cosas similares con nombres diferentes o poniendo diferente énfasis. No es la intención seleccionar uno a desmedro de otro. Diferentes técnicas resultan en diferentes puntos de vista del problema y se complementan entres sí. Por ejemplo, es necesario conocer sobre amenazas para definir defensas: utilizar patrones de ataque me puede facilitar la búsqueda de casos de mal uso. Algunas técnicas se enfocan en la elicitación, otras en análisis, y otras en la especificación, documentación, verificación o gestión de requerimientos a lo largo de todo el CVDS.

El portal de DHS Build Security In (https://buildsecurityin.us-cert.gov/bsi/home.html) incluye resultados de trabajos realizados (incluyendo casos de estudio) en el intento de definir un método que permita seleccionar una técnica entre varias disponibles. El criterio de selección incluye elementos tales como curva de aprendizaje y existencia de herramientas CASE que puedan dar soporte a la técnica [Mead 2006].

La adopción de una técnica sobre otra depende de la metodología de desarrollo de software utilizada. Algunas metodologías requieren técnicas que producen artefactos más fáciles de reutilizar o de incorporar en etapas posteriores del CVDS. En algunos casos, el proceso esencial definido en la metodología es el mismo, pero su implementación específica, terminología, notación, o rigurosidad requerida es diferente.

Modelar es una herramienta de aproximación de la ingeniería, que aplicada a la IR, permite descubrir y aprender sobre los requerimientos de software. Provee una forma de visualizar el trabajo y las iteraciones del producto propuesto en un ambiente entendido y controlado. Mientras más cercano sea el modelo a la realidad, más útil se torna a fin de entender la aproximación. De este modo, el desarrollo de software seguro se beneficia con aquellas aproximaciones que incorporan explícitamente artefactos del dominio de la seguridad como pueden ser las amenazas.

Los principales problemas cuando se modela son: 1) Hacer un buen modelo; 2) Disponer de criterios para mejorarlo; y 3) Saber qué hacer luego con los resultados; por ejemplo, cómo transferirlos a una métrica o a puntos que sean de utilidad para la toma de decisiones.

Hay varias técnicas y herramientas para modelar amenazas, ataques y vulnerabilidades. Microsoft, por ejemplo, ha reforzado este tipo de modelos en su iniciativa de software seguro. Por otro lado, en los últimos años han surgido múltiples metodologías que permiten a los desarrolladores conducir el modelado de amenazas y valoración de riesgos en la construcción de software.

2.4.4.1 Modelo de amenazas de Microsoft (STRIDE/DREAD)

El elemento central de los programas de Microsoft es el modelo de amenazas: una descripción textual y gráfica detallada de las amenazas significativas que permite al sistema de software ser modelado. Este modelo captura las formas mediante las cuales las funciones del software y su arquitectura pueden identificarse y ser objetivos de potenciales ataques; por ejemplo: vectores de ataque.

Para ayudar a definir los escenarios de amenazas, la Versión 1 del acrónimo STRIDE, le permite al usuario visualizar escenarios potenciales de amenazas, desde la perspectiva de un atacante, agrupándolas en categorías: (**S**poofing) robo de identidad; (**T**ampering) cambio no autorizado de

datos; (**R**epudiation) repudio de servicios o acciones; (**I**nformation disclosure) divulgación de información; (**D**enial of service) denegación de servicios y (**E**levation of privilege) elevación de privilegio. Luego utiliza una metodología de cálculo de riesgos conocida como DREAD, cuyo nombre encapsula las respuestas a potenciales preguntas sobre los riesgos:

- Daño potencial (**D**amage potential) : Qué tan grande es el daño si la vulnerabilidad es explotada..?
- Reproducibilidad (**R**eproducibility) : Qué tan fácil es reproducir el ataque..?
- Aprovechable (**E**xploitability) : Qué tan fácil es lanzar el ataque..?
- Usuarios afectados (**A**ffected users,) : En porcentaje, cuántos usuarios afectaría..?
- Descubrible (**D**iscoverability) : Qué tan fácil es encontrar la vulnerabilidad..?

DREAD ayuda a valorar las amenazas y priorizar la importancia de las contramedidas y acciones para mitigarlas. Una vez que los atributos de las amenazas son clasificadas, se toma una media de los cinco atributos, lo que resulta en una valoración del riesgo percibido asociado a la amenaza. Este proceso es repetido para todas las amenazas identificadas para luego priorizarlas por orden descendente.

En marzo del 2006, la empresa hizo una revisión de su modelo y lo renombró como "*Microsoft Threat Analysis and Modeling*", el cual intenta ser una nueva metodología y proceso para modelar amenazas, más amigable para desarrolladores de software, arquitectos y "*stakeholders*", no expertos en el dominio de la seguridad. Eliminó STRIDE y DREAD y desplazó la perspectiva desde un punto de vista centrado en el atacante hacia otro centrado en el defensor. El usuario se identifica estrechamente con las amenazas en lugar de hacerlo con los ataques, reflejando la visión de que el defensor debe tener un mejor conocimiento de las amenazas al sistema que el atacante. Se trata de un proceso iterativo, que va sumando capas de detalle a un modelo inicial de alto nivel, a medida que el diseño progresa sobre las fases siguientes del CVDS. Una amenaza se define como un evento con impacto negativo en los objetivos del negocio o misión de la organización. El nuevo modelo intenta aclarar la distinción entre amenazas, ataques y vulnerabilidades. Incorpora un catálogo de ataques predefinidos, describiendo las contramedidas asociadas para minimizar los efectos, auto-generando modelos de amenazas en base a contexto definidos para la aplicación. Luego el modelo mapea aquellas amenazas sobre las contramedidas.

2.4.4.2 Casos de uso, casos de mal uso y casos de abuso

La adaptación de los casos de uso es un conjunto de aproximaciones muy prometedor para el desarrollo de requerimientos que enfocan la seguridad del software. Les han dado varios nombres para distinguirlos del caso de uso estándar: casos de abuso, casos de mal uso, casos de uso hostiles, y casos de fiabilidad (enfocados en las excepciones). Lo que tienen en común todos ellos es que ven el software desde el punto de vista de un adversario, y si bien hay diferencias en los detalles, para los fines de este trabajo no las vamos a tener en cuenta.

Del mismo modo que los casos de uso se utilizan exitosamente para elicitar requerimientos, los casos de mal uso son utilizados para identificar potenciales amenazas, a partir de las cuales es posible elicitar requerimientos de seguridad o casos de uso de seguridad. La interacción del usuario autorizado con el sistema es diagramada simultáneamente con las interacciones del usuario hostil. Y así como en los casos de uso las conexiones entre actor y acción se etiquetan con términos como "extiende" e "incluye", las conexiones en un caso de mal uso son etiquetadas con "amenaza" y "mitiga". Los casos de mal uso forman los cimientos para construir un conjunto de casos de uso seguros para contrarrestar cada una de las amenazas. Como su caso de uso, cada caso de mal uso conduce un requerimiento y el correspondiente escenario de prueba para el software. De este modo,

gran parte del trabajo invertido en la construcción de los casos de mal uso resulta en el desarrollo de requerimientos funcionales de seguridad. No obstante, la técnica de análisis, provee una forma de elicitar requerimientos no funcionales para proteger el software en sí mismo.

La obra [Hope 2004] es una muy buena introducción al tema. El artículo hace una distinción entre casos de mal uso y casos de abuso, observando la "no intencionalidad" en la ocurrencia de los primeros y no así en los segundos, y por ellos mismo considerados hostiles. Es una distinción similar a la existente entre los riesgos y amenazas, pero no es una distinción estándar. También sugiere que deben ser empleado los patrones de ataque, para ayudar a identificar casos de mal uso. Esta tarea debe ser realizada equipos donde participan tanto desarrolladores de software (expertos en la materia) y expertos en seguridad.

Para una descripción concisa de la técnica, se puede consultar [Damodaran 2006]. De esta obra es el siguiente párrafo:

> *Descripción esencial de cómo construir casos de mal uso: para cada caso de uso, hacer una tormenta de ideas para identificar el modo en que los agentes negativos intentarán impedir o frustrar algunos de los pasos en la descripción del caso de uso; esto lleva a los principales casos de mal uso. Durante la sesión, debe ponerse el foco de atención en identificar las múltiples formas en que un atacante podría causar daño en el caso de uso bajo estudio; luego podrán completarse mas detalles del ataque. Cada uno de estos modos de ataque es un candidato firme a un caso de mal uso.*
>
> *El objetivo es identificar amenazas a la seguridad en cada una de las funciones, áreas, procesos, datos y transacciones involucradas en el caso de uso a partir de potenciales riesgos, como pueden ser los accesos no autorizados desde adentro y desde afuera; ataques de DoS; violaciones a la privacidad, confidencialidad e integridad y ataques malintencionados por parte de hackers. Además de estudiar los modos de ataque, el proceso podría también intentar descubrir posibles errores de los usuarios y la correspondiente respuesta de la aplicación. Frecuentemente estos errores podrían causar errores serios en la funcionalidad o en la seguridad de la aplicación. Identificando todas las acciones inapropiadas que podrían ocurrir, capturamos todas las acciones de uso anormal de la aplicación por parte de usuarios genuinos, en términos de accidentes o errores por descuido, o por parte de atacantes intentando romper o dañar el correcto funcionamiento de la aplicación.*

2.4.4.3 Árbol de ataque, árbol de amenazas y grafo de ataque

Árbol de ataque, árbol de amenazas y grafo de ataque son representaciones de un posible ataque en contra de un objetivo y ayudan a visualizar una serie de eventos más complejos (patrones de ataque) que pueden ser combinados para comprometer la seguridad de una aplicación. Todos ellos proveen formatos alternativos para capturar información de casos de abuso y casos de mal uso. La visión integral que proveen los árboles y grafos aportan claridad a las dependencias existentes entre patrones de ataque que explotan vulnerabilidades de software y aquellas que tienen como meta vulnerabilidades de otros niveles, como pueden ser la red, el personal o los procedimientos. Este punto de vista puede incrementar el conocimiento para la gestión de riesgos, para mejorar la coordinación de contramedidas y minimizar las vulnerabilidades a través de todas las capas del sistema.

El siguiente ejemplo de árbol de ataque, en una versión no gráfica, incluye varios patrones de ataque que explotan vulnerabilidades de software:

Objetivo: Hacer una reserva aérea falsa.

1. Persuadir al empleado de sumar una reserva.
   1.1. Chantajear empleado.
   1.2. Amenazar empleado.
2. Acceder y modificar la base de vuelos.
   2.1. Realizar una injección SQL desde el sitio web (V1).
   2.2. Logearse a la base de datos.
      2.2.1. Adivinar la clave.
      2.2.2. Espiar la clave (V7).
      2.2.3. Robar la clave del servidor web (AND).
         2.2.3.1. Crear una cuenta en el servidor web.
            2.2.3.1.1. Explotar un "buffer overflow" (V2).
            2.2.3.1.2. Obtener acceso a una cuenta de empleado.
         2.2.3.2. Explotar un "estado de corrida" para acceder a un archivo protegido (V3).

En la Figura 2.4.3 se muestra un grafo de ataque que incluye varios patrones de ataque que explotan vulnerabilidades de software.

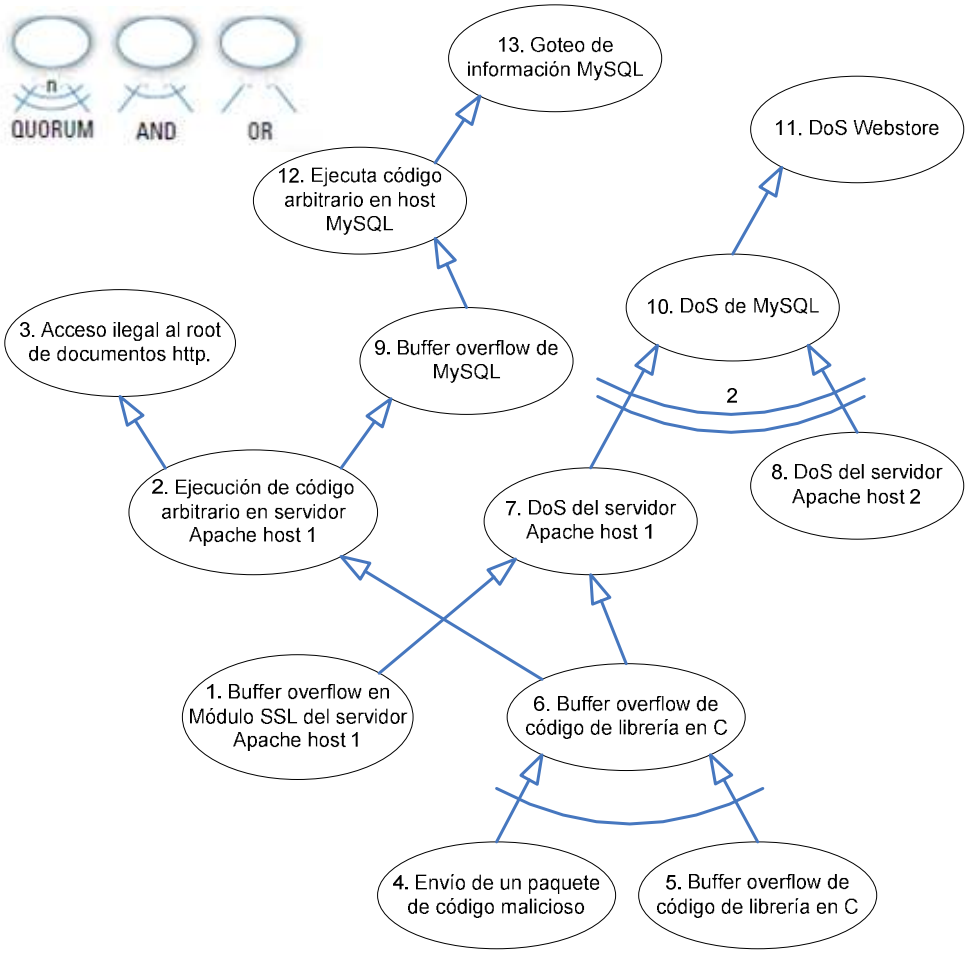

Figura 2.4.3. – Ejemplo de Grafo de Ataque

No todos los árboles de ataque y de amenazas, y grafos de ataque se prestan para un análisis manual, debido a su tamaño y complejidad. Algunos de ellos han sido generados para representar ataques distribuidos del mundo real, sobre grandes sistemas distribuidos; estos artefactos incluyen cientos y hasta miles de ramas y caminos simultáneos diferentes para completar el ataque.

De acuerdo a quien intente utilizar árboles de ataque y amenazas, pueden resultar difícil, sino imposibles de utilizar por usuarios no expertos en seguridad. Un árbol es una "lista de precondiciones relacionadas de seguridad", por lo que resulta poco realista esperar que un no experto en seguridad pueda generar dicha lista.

Hay empresas de desarrollo como Microsoft que han descubierto que estas listas de precondiciones relacionadas de seguridad laboriosamente generadas (árboles de ataque y amenazas) en realidad forman patrones que pueden ser estandarizados y reutilizados y que luego son fácilmente comprensibles para desarrolladores no expertos en seguridad. Esta aproximación mediante la utilización de patrones para modelar los ataques y amenazas a la seguridad, han desplazado los modelos que utilizan árboles de ataque y amenazas y grafos.

2.5 "*Refactoring*" Seguro

El "*refactoring*" de software es útil para mejorar características de calidad de software existente. De acuerdo a su definición, ampliamente aceptada en la comunidad de ingeniería de software, su principal objetivo es hacer el código más entendible y fácil de modificar, alterando su estructura interna, sin modificar su comportamiento externo observable. Pero, desafortunadamente, en la definición no hay ninguna referencia sobre aspectos referidos a la seguridad del software. A pesar de ello, revisando la bibliografía del tema, el personal involucrado con la tarea de implementación, puede encontrar algunas pocas referencias a cambios en la estructura interna de un software, que hacen el código más seguro, sin cambiar su comportamiento externo.

Concretamente, algunas recomendaciones propuestas de "*refactoring*", en el catálogo de Fowler [Fowler, 1999] pueden incrementar el nivel de seguridad del código existente. Por ejemplo, la recomendación de refactorización "*Encapsulate Field*" convierte el privilegio de accesibilidad de "público" a "privado" y suma "*accesors*" (seters y getters) al campo. Si un campo es declarado público, cualquier cliente (atacante en muchos casos) puede obtener fácilmente acceso a modificar el valor del campo. Esta transformación dificulta el acceso al campo encapsulado, de modo que se incrementa en nivel de seguridad del código resultante.

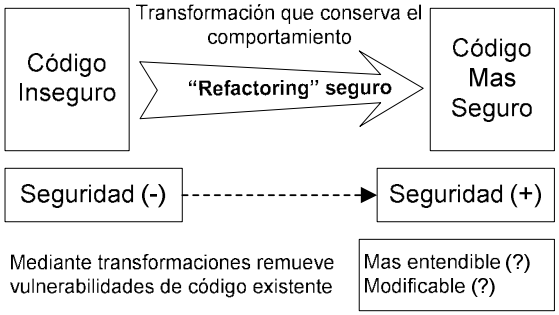

Figura 2.4.4. – Aspecto general del "Refactoring" seguro

El "*refactoring*" seguro [Maruyama 2007] es un conjunto de transformaciones seguras que incrementan el nivel de seguridad del código así tratado (ver Figura 2.4.4). Todas estas transformaciones garantizan que el comportamiento externo observable del código resultante no

cambie. Esto significa que el conjunto resultante de valores de salida para el mismo conjunto de valores de entrada legal (no malicioso) es el mismo antes y después de la transformación. La definición de "*refactoring*" seguro es la siguiente:

> *"Un cambio hecho en la estructura interna del software para lograr que sea más seguro, sin cambiar su comportamiento observable"*

Tanto el "*refactoring*" tradicional como el seguro tienen la misma propiedad de conservar el comportamiento externo después de una serie de transformaciones. La diferencia entre ellos es que el "*refactoring*" seguro se entiende que tiene como objetivo mejorar la seguridad de código existente, mientras que el tradicional se enfoca en la mantenibilidad del código. Obviamente, un código más entendible o modificable es útil para mejorar sus propiedades de seguridad ya que facilita la identificación de vulnerabilidades y su resolución. No obstante, muchos programadores con conocimiento de la seguridad del software frecuentemente encuentran situaciones en donde rápidamente pueden eliminar errores y defectos en el código sin tener en cuenta la mantenibilidad. El "*refactoring*" seguro está pensado para ser utilizado en estas situaciones. Empleando esta técnica en el desarrollo y mantenimiento de software el programador estará en condiciones de obtener software que es más seguro que el que se tenía antes, sin recurrir a tediosos chequeos.

Cada "*refactoring*" tiene un nombre, una motivación, un mecanismo y ejemplos. La motivación describe el por qué debe ser aplicado el "*refactoring*" y los resultados de su aplicación. El mecanismo muestra una situación en la cual el "*refactoring*" debería ser aplicado (llamado "*security smell*") y presenta un conjunto de procedimientos concisos de cómo llevarlo a cabo.

Desafortunadamente aún hay pocos estudios sobre el "*refactoring*" aplicado sobre aspectos de seguridad [Maruyama 2007] [Smith and Thober, 2006]. Incluso en bibliografía como [Kerievsky 2005], donde se plantea la posibilidad del "*refactoring*" en la dirección de los patrones de diseño, no hay ninguna mención al tema de la seguridad.

Por otro lado y como se dijo arriba, si bien muchos programadores remueven vulnerabilidades fácilmente de un código existente, aún hay pocos mecanismos sistemáticos que puedan ser automatizados en una herramienta. Lo cual es considerado un aspecto crucial. Sin dejar de tener en cuenta que es necesario que la herramienta pueda generar una implementación transformada que pueda correr para demostrar que el "*refactoring*" seguro es factible de ser aplicado.

Esta es una de las áreas que dará muy buenos resultados cuando puedan unirse los esfuerzos hechos en la dirección del "*refactoring*" con los Patrones de Seguridad, dada la cantidad de código existente inseguro en el mercado.

2.6 Modelos orientados a objetos y modelos orientados a aspectos

Los modelos y análisis orientados a objetos han traído beneficios desde la introducción de la notación estándar UML. Lo que en algún momento fue una serie de técnicas en conflicto, se ha transformado en prácticas comunes y aceptadas. La primera y principal consecuencia es el valor del modelo para tratar con la complejidad. En MDA (Model Driven Arquitecture) y MDD (Model Driven Development), los modelos UML son el artefacto principal de las fases de arquitectura y diseño de software y utilizando herramientas automatizadas, es posible traducir estos modelos en código.

MDA está siendo utilizado en el desarrollo de software seguro en el ámbito comercial. Los objetos de seguridad hacen posible una Secure MDA, la que provee una arquitectura consistente para modelar, generar y hace cumplir políticas de seguridad, mediante MDA. La organización Object Security ha desarrollado una aplicación que integra un SecureMiddleware utilizando CORBA

(Common Object Request Broker Architecture) del OMG. Las políticas de seguridad generadas por la Secure MDA son impulsadas por el nivel de middleware, previniendo de este modo que cualquier componente malicioso o impropio afecte el resto del sistema distribuido. Si bien Secure MDA sólo hace cumplir modelos de seguridad a nivel de la capa de middleware, es un paso importante poder utilizar modelos para desarrollar sistemas con un nivel de seguridad alto.

Por otro lado, los modelos orientados a aspectos (Aspect Oriented Modeling, AOM), la primera fase de un desarrollo de software orientado a aspectos (AOSD), son una técnica MDD que se enfoca específicamente sobre la dificultad de capturar y describir propiedades transversales tales como la seguridad. AOM separa el diseño de los aspectos no funcionales de diseño de los aspectos funcionales; luego compone los modelos separados para crear el modelo de una solución integral. Es habitual construir varios modelos, cada uno con un propósito específico. Un primer modelo incorpora una solución que reúne los requerimientos funcionales más importantes. Otros modelos incorporarán soluciones para requerimientos funcionales como la seguridad o aspectos de la tolerancia a fallas. Estos modelos, de aspectos del software, se integran con el primer modelo de acuerdo a un conjunto de reglas de composición, para producir un modelo integral de la solución propuesta.

Los perfiles de seguridad para UML, de dominio público, son los siguientes:

- **UMLSec [UMLSec]**: El grupo de trabajo Security and Safety in Software Engineering de la Technical University of Munich desarrolló esta propuesta y una extensión de la herramienta CASE AutoFocus que tiene en cuenta información de seguridad. Ambas herramientas pueden ser utilizadas de forma conjunta para verificar requerimientos de seguridad. Dado que UMLSec está basado en un estándar, se considera que es una herramienta para desarrolladores no expertos en seguridad.
- **SecureUML [Basin 2006]:** Creado por Torsten Lodderstedt, David Basin, y Jürgen Doser del Institute for Computer Science en University of Freiburg (Germany) y aplicado por Foundstone en el diseño de sistemas de autorización seguros, SecureUML es un lenguaje de modelado basado en UML que expresa el control de accesos basado en roles (RBAC) junto a restricciones de autorización, en el diseño general de una aplicación. Si bien UML es factible e utilizarse para modelar muchas otras propiedades de la seguridad del software tales como integridad, disponibilidad, etc., SecureUML se enfoca en la autorización y control de acceso.
- **Perfiles UML de CORAS [CORAS 2011]:** Desarrollado por CORAS Project, estos perfiles proveen el meta-modelo que define el lenguaje abstracto para modelar amenazas tales como explotación de un “buffer overflow”, así como las contramedidas o tratamientos asociados. El perfil permite mapear clases en el meta-modelo para modelar elementos con el estándar UML. Fue recomendado como un estándar por el OMG en noviembre del 2003.

El modelo orientado a aspectos (AOM) está siendo adoptado para capturar requerimientos de seguridad de software. Los aspectos son una construcción que permite capturar no solo el comportamiento del sistema. También características no funcionales, referidas como propiedades transversales, las que no pueden ser representadas de manera efectiva con modelos orientados a las funcionalidades. Algunos ejemplos frecuentemente citados de tales aspectos, incluyen la sincronización, interacción de componentes, persistencia, tolerancia a fallas, calidad de servicio, dependencias y seguridad. Se sobreentiende que AOM es usada, conjuntamente con modelos UML orientados a la funcionalidad (con o sin extensiones tales como UMLSec o SecureUML), para mapear aspectos sobre entidades del modelo de transformación.

2.7 Conclusión sobre el estado del arte del aseguramiento del software

El estado del arte de la seguridad del software muestra esfuerzos hechos para encontrar consenso en el uso de diferentes técnicas y herramientas que garanticen una propiedad del software que pareciera no es posible garantizar en un 100%. Pero de esos esfuerzos se pueden rescatar hitos claramente vinculados con este trabajo de tesis:

a) Los requerimientos de seguridad están ausentes o pobremente especificados debido a un inadecuado entendimiento de la seguridad en general y de la seguridad como una propiedad particular, por parte de la mayoría de los profesionales que hacen requerimientos.
b) La selección de los requerimientos de seguridad es inconsistente con la mayoría de los requerimientos funcionales. A partir de tener en cuenta los requerimientos de seguridad se suman inconsistencias debido a que mientras que algunos requerimientos son incluidos, otros de los cuales dependen, son dejados de lado.
c) La necesidad de incorporar la seguridad en todas las etapas del CVDS, incluyendo y recomendando fuertemente esta tarea en la elicitación de requerimientos.
d) Las metodologías ágiles están intentado desarrollar aspectos sobre qué tratamiento darle a la generación de software seguro.

Por otro lado, modelar requerimientos de seguridad permite descubrir y aprender sobre los requerimientos de software, resolver posibles conflictos e identificar amenazas. Provee una forma de visualizar el trabajo y las iteraciones del producto propuesto en un ambiente entendido y controlado. Mientras más cercano sea el modelo de requerimientos de seguridad a la realidad, más útil se torna a fin de entender la aproximación y proponer alternativas para minimizar su vulnerabilidad. De este modo, el desarrollo de software seguro se beneficia con aquellas aproximaciones o modelos, que incorporen explícitamente artefactos del dominio de la seguridad (Patrones de Seguridad).

3 Léxico Extendido del Lenguaje (LEL), Escenarios y Patrones de Seguridad

Los conceptos de Antonelli [Antonelli 2003] en particular para la generación de software seguro, siguen siendo válidos. La investigación en el desarrollo de software aún centra sus esfuerzos en obtener nuevas técnicas y metodologías para construir software, pero los problemas vinculados a extraer del dominio específico la solución: construir requerimientos, sigue siendo un problema no fácil de resolver. A esto se suma que en todo momento se está atacando el problema mirando puntualmente los requerimientos funcionales, dejando afuera los requerimientos no funcionales, entre ellos la seguridad. Una estrategia como esta puede ser de utilidad cuando los problemas que se atacan son bien conocidos. Sin embargo, esta conclusión no es válida para el desarrollo de software con requerimientos de seguridad, en donde el análisis del problema puntual de elicitación de requerimientos no está definitivamente resuelto.

Es necesario construir un modelo contextual que permita incorporar el dominio de la seguridad. Para construir este modelo, es imprescindible utilizar una técnica que permita su uso por parte de no expertos en el dominio de la seguridad. Dos herramientas que han mostrado ser adecuadas y efectivas para la elicitación de requerimientos en otros dominios, son el modelo de LEL y escenarios, los cuales se describen en las secciones 3.1 y 3.2, a la cual se le suman los casos de uso como representaciones gráficas previas a la construcción de escenarios. A este modelo se le agregan las tarjetas CRC descriptas en la sección 3.3. Esto permite contar con un modelo integrado que comienza en el modelo contextual y abarca hasta el diseño preliminar de objetos. Las relaciones entre LEL, escenarios y tarjetas CRC se describen brevemente en la sección 3.4. En la Sección 3.5 se presentan los Patrones de Seguridad, con dos ejemplos: "*Packet Filter Firewall*" y "*Generic Object-Oriented Cryptographic Architecture*" (GOOCA). En 3.6 se muestra que es perfectamente posible modelar Patrones de Seguridad con LEL y Escenarios.

3.1 Léxico Extendido del Lenguaje

El Léxico Extendido del Lenguaje (LEL) es un modelo contextual que permite capturar el lenguaje de un dominio. Identificando y definiendo los símbolos propios de un contexto se logra un mejor entendimiento de éste. La idea bajo el LEL es muy simple: primero hay que preocuparse por entender el lenguaje del problema, sin preocuparse por entender el problema. El LEL es un glosario que permite registrar signos (palabras o frases) los cuales son específicos del dominio. Cada signo del LEL es descripto de dos formas: a través de la noción y los impactos.

La noción es la descripción del tipo usual que da el diccionario. Es la descripción del signo por medio de sus propiedades intrínsecas. Mientras que los impactos determinan cómo el signo descripto se relaciona con los demás. Los impactos describen la incidencia del signo en los demás o la de los demás en el signo.

Dos principios rigen la construcción del LEL: circularidad y vocabulario mínimo. El primero establece que en la descripción de la noción e impactos se debe maximizar el uso de signos definidos en el LEL. El segundo complementa el principio de circularidad, y establece que al utilizar signos externos al LEL, deben tener una representación matemática clara (por ejemplo: pertenencia, intersección, etc). Ambos principios logran que el conjunto de signos sea auto-contenido: que un signo esté expresado en término de otros. De esta forma los signos están interrelacionados. Luego si cada signo se ve como un nodo de información, junto con la relación entre ellos se puede percibir un grafo y como cada nodo está conformado por texto, el grafo no es otra cosa que un hipertexto [Leite 1997].

3.1.1 Proceso de construcción del LEL

La construcción de LEL comienza por obtener información del dominio. A partir de esta información se elabora una lista de símbolos que se deben conocer para entender su lenguaje. Estos símbolos se deben clasificar para poder definirlos en forma consistente. Luego de clasificarlos, se los define y como producto de la definición se pueden descubrir sinónimos, por lo cual se deben reorganizar los símbolos. La información debe ser validada por los expertos del dominio y controlada por el ingeniero de requerimientos. Si algún nuevo símbolo debe ser definido, se repite el proceso.

El proceso consta de los siguientes pasos:

- **Identificar las fuentes de información:** Las fuentes de información para la construcción de LEL son dos: personas y documentos. Los documentos ofrecen un medio concreto, autocontenido e invariable. El ingeniero de requerimientos puede leer y releer los documentos para lograr una selección minuciosa de los términos. Por otra parte, las personas pueden aportar a través de entrevistas mucha mayor información. Una característica propia de la expresión oral de los seres humanos es que en forma inconsciente utilizan el principio de circularidad. En cambio en la descripción escrita, para una narrativa más entretenida se introducen muchos sinónimos, lo que puede dificultar el proceso de identificación de símbolos. Es aconsejable utilizar ambas fuentes de información.
- **Generar la lista de símbolos:** Para generar la lista de símbolos es aconsejable comenzar con una lectura de la documentación para hacer un análisis preliminar del lenguaje del dominio. Hecho este análisis se obtiene una lista inicial y se adquiere información para llevar a cabo entrevistas. Es aconsejable realizar entrevistas semi-estructuradas y estructuradas para acotar el lenguaje. Si se comienza con entrevistas libres, el volumen de información es difícil de manejar. Los símbolos que se deben tomar para la lista inicial son las palabras o frases que se utilizan con mucha frecuencia, o aquellas que parecen estar fuera de contexto. El motivo de elegir las palabras o frases que se utilizan con mucha frecuencia es claro. El objetivo del LEL es capturar el lenguaje de un dominio. Los términos más utilizados son significativos en el dominio, por lo tanto deben estar definidos. En cambio los términos que parecen estar fuera de contexto, se deben a que tienen un significado propio en el dominio, distinto del tradicional. Estos términos, con más razón deben estar definidos. Cabe destacar que los símbolos no tienen porque ser palabras individuales. Pueden ser palabras o frases. La razón es que en un lenguaje se puede encontrar un grupo de palabras, en donde cada una de ellas pueda tener cierto significado, pero si se las combina en una frase tienen otro significado. Si bien en la etapa de definición es cuando se encuentran los sinónimos, la tarea puede comenzar en esta etapa.
- **Clasificar la lista de símbolos:** El objetivo de categorizar los símbolos es lograr una mejor administración del conjunto. Cada categoría determina la forma en que se debe definir cada símbolo y de este modo se logra una definición consistente y uniforme. Para clasificar los símbolos se parte de una clasificación general:
    - **Sujeto:** Elemento activo del dominio que realiza acciones utilizando objetos. Puede llegar a pasar por distintos estados.
    - **Verbo:** Acción que realiza un sujeto, servicio que brinda un objeto, desencadenante para pasar de un estado a otro.
    - **Objeto:** Elemento pasivo con los cuales se realizan acciones que puede pasar por distintos estados.

- **Estado:** Situación en la que se encuentra un sujeto o un objeto.

Esta es una clasificación inicial que en función del dominio se puede especializar. La especialización puede darse por ejemplo en los sujetos. Un sistema de administración de empleados de una empresa tiene distintos tipos de empleados. Aquellos en actividad, en condición de retiro y retirados. Y estos a su vez pueden contener sub-categorías, por lo cual se necesita explotar la categoría inicial. Esta categorización permite agrupar símbolos relacionados y después encontrar sinónimos.

Para cada categoría se debe definir su forma de describir la noción e impacto de los símbolos. Así se asegura que los símbolos son descriptos consistentemente y que es posible contrastarlos, para un mejor entendimiento del lenguaje. Algunas pautas a tener en cuenta en la descripción de los símbolos son las siguientes:

- Para los signos que son sujetos, los impactos deben indicar las acciones que realiza.
- Para los signos que son verbo, las nociones deben decir quién ejecuta la acción, cuándo sucede y el proceso involucrado en la acción. Para los impactos se deben identificar las restricciones sobre la realización de la acción, qué origina esta acción y qué es lo que causa esta acción.
- Para los signos que son objetos, la noción debe identificar otros objetos con los cuales se relaciona. Y los impactos serán las acciones que se pueden realizar con este signo.

Estas reglas deben ser extendidas en la medida que la categorización inicial lo sea.

- **Describir los símbolos:** Los símbolos se describen a partir del conocimiento obtenido de la lectura de la documentación y de las entrevistas. Cada símbolo se describe siguiendo las pautas establecidas en la clasificación de los términos. Esta descripción inicial luego será validada y controlada. Algunas pautas generales, independientemente de la clasificación a la que correspondan, son las siguientes:
  - Un signo puede tener una o más nociones y cero o más impactos.
  - Cada noción e impacto debe ser descripto con oraciones breves y simples.
  - Las oraciones deben responder a los principios de circularidad y de vocabulario mínimo.
  - Noción e impacto de un signo pueden representar diferentes puntos de vista o pueden ser complementarios.

  Con la descripción de símbolos se facilita la tarea de encontrar sinónimos. Sin embargo, en la validación posterior, se despeja cualquier duda respecto de los sinónimos.
- **Validar:** Consiste en verificar la correctitud del LEL contra el usuario. Debido a la gran cantidad de símbolos que pueden estar definidos en el LEL, es impracticable realizar una validación completa y exhaustiva. Sin embargo, es posible mantener sesiones de entrevistas estructuradas para aclarar dudas.
- **Controlar:** El proceso de control lo realiza el ingeniero de requerimientos por sus propios medios. No tiene que ver con la correctitud de la información del LEL. Está relacionado con la estructura:
  - Todos los símbolos deben estar definidos.
  - Todos los símbolos deben estar dentro de la clasificación correspondiente.
  - La descripción de los símbolos debe corresponderse con la de la categoría a la que pertenecen.

- El punto de vista para la descripción de los símbolos debe ser uniforme.
- No deben dejarse sinónimos definidos como signos independientes.

3.2 Modelo de escenarios

Los aspectos dinámicos del dominio de la aplicación se capturan mediante escenarios, a los que se considera como una representación de casos de uso en el contexto de historias y objetivos que permiten mostrar con claridad los límites de la aplicación. Los casos de uso son una técnica para la obtención de requerimientos y en este trabajo se consideran directamente mapeables sobre un escenario en los términos que se describe a este último a continuación.

Las características principales de los escenarios son las siguientes:

- Representan situaciones con énfasis en la descripción del comportamiento.
- Utiliza la descripción textual como representación básica, lo cual está en un todo de acuerdo con el mapeo de casos de uso con escenarios.
- Está naturalmente ligado al LEL al describirse con el vocabulario definido en el este último.

Los atributos que definen a un escenario son: título, objetivo, contexto, recursos, actores y episodios. Título, objetivo, contexto, recursos y actores son oraciones declarativas, mientras que episodios es un conjunto de oraciones de acuerdo con un lenguaje muy simple que hace posible la descripción operacional del comportamiento. Las características de cada atributo son:

- **Título:** Identifica al escenario. En el caso de un sub-escenario el título es el mismo que la oración del episodio desde el cual es referenciado.
- **Objetivo:** Meta a ser alcanzada.
- **Contexto:** Describe el estado inicial del escenario.
- **Recursos:** Elementos con los cuales se llevará a cabo el escenario.
- **Actor:** Persona o objeto que impulsa el escenario.
- **Episodios:** Se forman con una serie de oraciones que detallan el comportamiento del escenario. Cada oración indica una tarea en la secuencia del escenario. Hay un par de símbolos especiales. El símbolo "#" se utiliza para indicar acciones no secuenciales. Cuando no importa la secuencia en un grupo de pasos se los debe encerrar entre "#". Para indicar tareas condicionales se utiliza if then con la semántica tradicional.

Los atributos contexto, recursos y episodios poseen modificadores como ser: restricciones y excepciones. Restricciones indican condicionantes. Por ejemplo, un recurso no puede usarse en tal situación. Las excepciones plantean casos alternativos. Por ejemplo los episodios se desarrollan siempre en cierta secuencia, sin embargo, excepcionalmente pueden llevarse a cabo otras tareas. En la figura 3.2.1 se muestra un diagrama E-R con la estructura de un escenario.

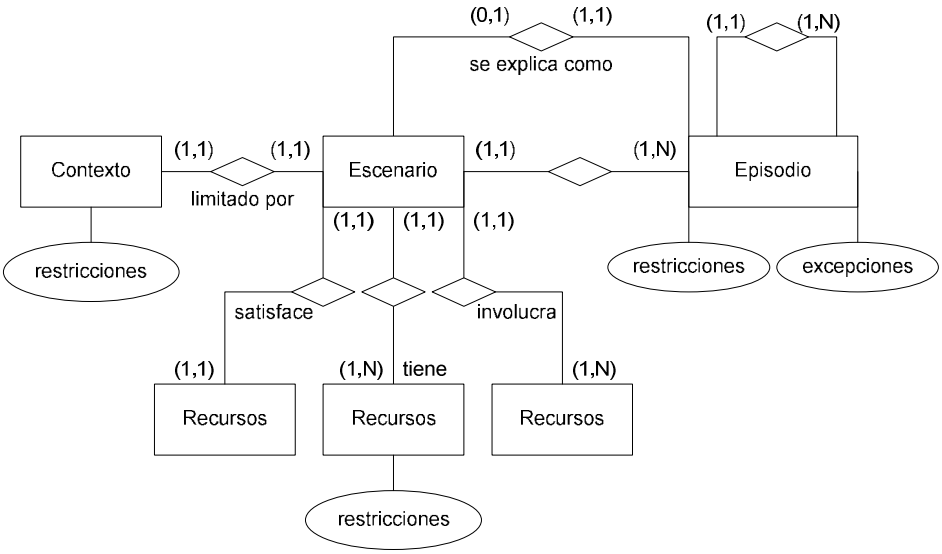

Figura 3.2.1. – Diagrama entidad relación de la estructura de un escenario

De la misma forma que en el LEL los símbolos se relacionan entre ellos dando, los escenarios también se relacionan entre ellos y con el LEL. Existe una relación estructural entre los escenarios. Un escenario puede ser subescenario de otro. Es decir, en un episodio puede aparecer una tarea lo suficientemente compleja, como para necesitar una explicación en calidad de otro escenario. Los escenarios se describen con texto que está definido en el LEL. Por lo tanto, los escenarios se ligan naturalmente a los símbolos del LEL.

3.2.1 Proceso de construcción de escenarios

El proceso de construcción de escenarios, en este trabajo, toma como punto de partida la definición de los casos de uso y al LEL. Se toman los símbolos que caen dentro de la categoría de sujetos y se toman sus impactos como potenciales escenarios. Una vez descriptos los escenarios se deben revisar y validar. El proceso es el siguiente [Antonelli 2003]:

- **Identificar los sujetos:** Los símbolos de LEL que caen en la categoría de sujetos corresponden a entidades activas que realizan tareas dentro del dominio. Los impactos del LEL de un símbolo sujeto deben capturar las acciones que éste realiza. Estas acciones son candidatas a convertirse en escenarios. En esta etapa hay que concentrarse en tomar todos los símbolos que caen en la categoría sujeto o en alguna categoría derivada de sujeto si es que se extendió. Luego hay que dividir la lista en actores principales y secundarios. La calificación de principal y secundario lo determina la relevancia del sujeto dentro del dominio de la aplicación. Esta categorización tiene como fin una mejor organización de los escenarios.
- **Generar lista de escenarios candidatos primarios:** A partir de los actores principales se extraen del LEL sus impactos. Cada impacto es un escenario potencial. Se nombra cada escenario con la acción en infinitivo contenida en el impacto. De esta lista inicial hay que eliminar los escenarios repetidos, ya que distintos actores pueden realizar la misma acción.
- **Describir los escenarios candidatos primarios:** A partir de la información de la entrada de LEL se pueden cubrir todos los atributos del escenario. El objetivo del escenario está

muy relacionado con el título. Es básicamente cumplir con la acción enunciada en el título. En el contexto se deben indicar las precondiciones y la situación espacio-temporal en que desarrolla el escenario. Esta información se puede extraer de los impactos del sujeto que originó el escenario. Los impactos previos al del escenario en cuestión pueden dar pautas de precondiciones necesarias. Los recursos son objetos con los que se desarrolla el escenario. La lista de recursos se puede obtener de entradas de LEL de la categoría objeto (o sus derivados) referenciadas por el sujeto que originó el escenario. Actor es el sujeto del cual se extrajo de sus impactos el escenario candidato en cuestión. Si varios actores coinciden en el escenario, todos ellos deben integrar la lista. Los episodios se completan con una descripción para que el actor o actores, a partir del contexto inicial y utilizando los recursos logren el objetivo.

- **Ampliar la lista de escenario candidatos:** A partir de la lista de actores secundarios se realiza el mismo proceso que con los actores primarios.
- **Describir escenarios candidatos secundarios:** Se realiza de la misma forma que se hizo con los escenarios candidatos primarios.
- **Revisión de escenarios:** Consiste en factorizar el conjunto de escenarios. Se pueden obtener nuevos escenarios como producto de uno o más episodios dentro de un escenario. O inversamente, dos o más escenarios pueden tener episodios u objetivos comunes, por lo cual es conveniente unirlos.
- **Validación de los escenarios:** Consiste en confrontar contra el experto del dominio la información en los escenarios, al igual que se hace con el LEL.

3.3 Tarjetas de Colaboración y Responsabilidad de Clases (CRC)

A partir de LEL y escenarios es posible obtener tarjetas CRC (colaboraciones y responsabilidades de las clases). Estas tarjetas identifican posibles clases para diseñar una solución orientada a objetos. Cada tarjeta representa un objeto del mundo real. Las CRC deben su nombre a que cada tarjeta identifica una clase. En ella se identifican las acciones que realiza: sus responsabilidades. Las tarjetas CRC también registran las relaciones con otras clases: sus colaboraciones.

Muchos especialistas del paradigma orientado a objeto manifiestan que identificar objetos es una tarea simple e intuitiva, mientras otros sostienen que esto no siempre es verdadero, particularmente con aplicaciones de gran escala. La técnica de **role playing** ataca este problema y consiste básicamente en jugar con las tarjetas disponiéndolas sobre una mesa, de la misma forma que en el juego de **scrabel** se cambia el orden de las fichas para buscar nuevas palabras ([Wilkinson 1995] y [Bellin 1997]).

Las tarjetas CRC descriptas anteriormente sirven para elicitar conocimiento de un dominio y se las conoce como CRC de análisis. Estas tarjetas se pueden transformar en CRC de diseño y utilizarse en un diseño preliminar de objetos.

3.3.1 Reglas de derivación de tarjetas CRC

Leonardi propone una estrategia para obtener tarjetas CRC a partir de LEL y escenarios [Leonardi 2001]. La estrategia consiste básicamente de tres tareas. Identificar CRC primarias, identificar CRC secundarias y determinar colaboraciones entre todas las CRC.

- **Encontrar CRC primarias:** Las tarjetas CRC primarias representan objetos con comportamiento relevante dentro del dominio. Los actores de los escenarios son

candidatos a convertirse en CRC primarias ya que realizan las tareas del escenario para lograr su objetivo. Los actores son entradas de LEL de la categoría sujeto. Estos sujetos realizan acciones en el dominio, las que son descriptas en los impactos. Entonces, los impactos no son otra cosa más que los servicios que provee, estos servicios son las responsabilidades de la CRC. De esta forma la tarjeta CRC es obtenida de un actor de un escenario y sus responsabilidades son los impactos de la entrada del LEL.

- **Encontrar CRC secundarias:** CRC secundarias son aquellos colaboradores de las CRC primarias que las ayudan a cumplir con sus responsabilidades. Las CRC secundarias se encuentran en las responsabilidades de las CRC primarias. Aquellos términos de las responsabilidades que también están definidos en el LEL se convierten en CRC secundarias. Las responsabilidades se obtienen de la misma forma que se obtienen las responsabilidades de las CRC primarias.
- **Encontrar colaboraciones:** Si las CRC secundarias se obtienen a partir de las responsabilidades de las CRC primarias, es trivial que colaboran en alguna medida. Sin embargo, para completar las colaboraciones, es necesario analizar los escenarios. En los episodios de los escenarios se deben buscar las entradas de LEL que originan a las CRC tanto primarias como secundarias. Las tarjetas CRC que participan de un mismo escenario colaboran entre ellas.

3.4 Relación entre LEL, escenarios y tarjetas CRC

LEL, escenarios y tarjetas CRC con su heurística de derivación conforman un método que permite obtener el conocimiento necesario para construir un diseño preliminar de objetos de la realidad, partiendo del modelo del dominio de la aplicación.

Todos estos elementos están interrelacionados. El LEL brinda la materia prima para la construcción de los escenarios, y a su vez estos son una descripción en términos de historias y objetivos de los casos de uso. Luego, LEL y escenarios, utilizando las reglas de derivación, permiten obtener tarjetas CRC. Hasta esta instancia del análisis y modelado de requerimientos, los requerimientos de seguridad se insinúan en el modelo, con frases como “Encripta/Desencripta información” y “Genera nonce” (nonce: number used once), entre los impactos de los términos de LEL que luego darán lugar a la aparición de CRC Primarias, pero el modelo en sí no aporta una solución específica para su tratamiento.

Cualquier variación en las entradas del LEL o los escenarios puede determinar que de acuerdo a las heurísticas de derivación se presenten condiciones para que varíen las tarjetas CRC (que aparezcan nuevas, que dejen de existir o sufran modificaciones). La variación del LEL o de algún escenario es común debido a la naturaleza dinámica del dominio. Por lo tanto, se necesitan de técnicas para propagar en forma automática los cambios que se producen. Surge la necesidad de proveer forward traceability.

3.5 Patrones de Seguridad

3.5.1 Origen del concepto de patrón de diseño

En el año 2001, en un intento por poner en ridículo el sistema de patentes, un australiano patentó un "*…dispositivo circular para facilitar el transporte…*", en realidad se trataba de la rueda, que se había reinventado nuevamente, como tantas veces se ha hecho desde hace miles de años. En una época, fuertemente influenciado por las TICs, si bien las innovaciones son conocidas globalmente

de forma prácticamente instantánea, las pequeñas soluciones a problemas concretos siguen siendo olvidadas e inventadas de nuevo, una y otra vez.

Christopher Alexander nació el 1936 en Austria, aunque pasó sus primeros 22 años en Inglaterra, donde estudió arquitectura y matemáticas, mudándose finalmente a Estados Unidos donde actualmente ejerce como profesor emérito en la Universidad de California. Al mudarse empezó a aplicar sus conocimientos matemáticos y el racionalismo al diseño, culminando en la publicación del libro "*Notes on Synthesis of Form*" en 1964. Sin embargo, al cabo de unos años él mismo se corregía:

> *"Soy, como algunos sabréis, matemático de origen. Dediqué muchos años, en los sesenta, a intentar definir una visión del diseño aliada con la ciencia [...]. Jugué con la investigación de operaciones, la programación lineal, todos estos juguetes fascinantes, que los matemáticos y la ciencia nos ofrecían, e intenté ver cómo todo esto podía darnos una nueva visión del diseño, qué diseñar y cómo diseñarlo. Finalmente, sin embargo, observé que esta visión no era productiva."*

Christopher Alexander se dio cuenta que el diseño no era algo matemático, no podía realizarse sin tener en cuenta las sensaciones y estaba muy relacionado con la vida. El diseño debía responder a necesidades de personas: un banco en un parque no era un "*banco*", era "*el banco de un grupo de personas*". Los pequeños detalles de diseño, sea en un edificio o una ciudad, debían responder a multitud de necesidades relacionadas con su entorno y, a la vez, integrarse a ese entorno. Cada una de estas necesidades tenía un modelo de solución que podía ser imitado cada vez que se detectaba el mismo problema, fuera en otro momento o en otro espacio. Este modelo replicable al que se refería Alexander, no era ni más ni menos, que un patrón de diseño.

Christopher Alexander publica en el año 1977 su libro "*A Pattern Language*", donde se puede leer su teoría de cómo el lenguaje de patrones puede ayudar a entender los objetos no como objetos aislados en sí, sino como elementos de interacción humana. Él define un patrón de diseño como:

> "*…una descripción de un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar la solución un millón de veces, sin hacer lo mismo dos veces…*".

Esta descripción da un nombre al patrón; describe el problema que trata; ofrece una solución, y finalmente habla de las consecuencias, ventajas e inconvenientes, que tiene esta solución. Por ejemplo, uno de sus patrones de diseño de edificios, llamado originalmente "*South facing outdoors*", constata que la gente no usará un espacio abierto si no es soleado (exceptuando climas muy cálidos) y propone la solución de colocar siempre los espacios abiertos, como un jardín, en la parte sur del edificio. Para nuestro hemisferio sería al norte, pero la idea es totalmente aplicable.

Posteriormente los patrones fueron introducidos en el dominio del desarrollo de software por diferentes caminos. Algunos desarrolladores fueron inspirados directamente por los trabajo de Alexander. Otros utilizaron fuentes similares, redescubriendo y reinventando aproximaciones sobre los diseños que ya se conocían. Frecuentemente se pueden encontrar soluciones similares para resolver un problema que tiene en cuenta una funcionalidad en particular, que resisten el paso del tiempo y cuyas consecuencias son muy bien conocidas.

En el libro "*Design Patterns: Elements of Reusable Object-Oriented Software*", Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides documentaron problemas de diseño que son recurrentes y las soluciones habitualmente encontradas en los frameworks orientados a objetos, escritos en C++ y Smalltalk. James Coplien documentó idiomas comunes y avanzados para C++. Kent

Beck y Ward Cunningham aplicaron las ideas de Alexander directamente sobre el diseño de las interfaces de usuario en Smalltalk.

Los patrones de diseño de software proporcionan, en el diseño iterativo, las mismas ventajas que en cualquier otro ámbito y pueden ser muy útiles en cualquier proyecto ya que:

- Hacen más eficiente el diseño, pues no es necesario volver a generar alternativas para solucionar un problema y elegir una de ellas. Esto permite la reutilización del software.
- Por su antigüedad, ya han sido probados y se conoce su eficacia. Esto evita riesgos en el desarrollo de una nueva solución para un problema conocido.
- Permiten crear estándares de uso, los usuarios acaban conociéndolos e, incluso, esperándolos, lo que contribuye a un fácil aprendizaje de los sistemas.
- Crean un vocabulario común que contribuye a mejorar la comunicación interna entre diseñadores.
- Su aplicación es viable en distintas fases del proceso de diseño, desde la conceptualización hasta el diseño final, siempre y cuando se conozcan los requerimientos de usuario.

Debemos tener en cuenta en todo momento que los patrones son una herramienta de diseño entre muchas otras. De hecho, Christopher Alexander define en su libro "*The Oregon Experiment*" hasta seis principios que se deben seguir a la hora de diseñar, uno de los cuales es utilizar un lenguaje de patrones. El segundo, en lo que se refiere al diseño por interacciones, seguramente resulta familiar:

> *"El principio de participación: todas las decisiones sobre qué construir y cómo construirlo deben estar en manos de los usuarios."*

Lo cual me animo a trasladar en estos términos:

> *"Todas las decisiones sobre qué funcionalidad implementar en una aplicación y cómo construirla debe estar en manos de los requerimientos."*

Los patrones de diseño de software, actualmente están documentados y presentados en varios libros, artículos, sitios web (http://www.hillside.net/), conferencias "*Pattern Language Of Programs*" (PLOP) y cursos.

3.5.2 Introducción a los patrones de diseño de software

En la Ingeniería de Software, los patrones de diseño identifican y documentan técnicas y experiencias de diseño bien conocidas, de manera que sea más simple su reutilización. Retomando una de las motivaciones de la programación orientada a objetos: la de reutilizar código a fin de aumentar la productividad en el desarrollo de software, los patrones se proponen reutilizar ideas en lugar de código, como instancia superadora.

Ponen énfasis en diseños prácticos y probados en lugar de trabajos originales o recién inventados, lo que aporta confianza a la toma de decisiones en las etapas de diseño. En lugar de reinventar soluciones conocidas, nos dan libertad para concentrarnos en las nuevas características y desafíos de la aplicación que se desea construir partiendo de una abstracción conocida.

Los patrones de diseño de software capturan soluciones recurrentes a problemas similares en un contexto dado, con sus ventajas y desventajas. No sólo son la forma de la solución sino que aportan conocimiento sobre el contexto sobre el cual se pueden utilizar y las consecuencias de su utilización. Un diseño o un estilo de diseño que trabaja correctamente en un contexto puede no ser completamente apropiado en otro, y hasta en ocasiones puede ser contraproducente. Los patrones también pueden ser

utilizados de forma inapropiada, cuando los desarrolladores no observan el contexto de aplicabilidad. Asumen que el patrón se aplica universalmente y no es así. Por ejemplo, una recomendación habitual sobre el diseño de métodos, es que un método debe tener un propósito simple y bien definido. Esto trabaja bien para una programación secuencial y local. Pero puede conducir a generar problemas si es mal adaptada sobre ambientes multi-hilo y distribuidos. La combinación de métodos múltiples dentro de un contexto único es una solución más apropiada en tales situaciones.

Dado que los patrones capturan el conocimiento y las estructuras en los elementos de un sistema, se los puede utilizar para entender y documentar estructuras existentes. Son guías para comprender aplicaciones, frameworks y librerías. Y debido a que capturan diseños comunes y repetidos, muchas veces son re-inventados o utilizados por hábito en el desarrollo de sistemas de software.

Los patrones ayudan a influir en las decisiones de diseño y a impulsar el diseño hacia adelante en el proceso de desarrollo, permitiendo hacer consideraciones sobre aproximaciones existentes y probadas. Los patrones capturan experiencias de diseño que se introducen directamente en soluciones para problemas puntuales de desarrollo. Por otro lado, en la comunicación de los diseños, los patrones ayudan a los desarrolladores a comunicar y justificar decisiones relevantes. Esto permite una rápida comunicación entre ellos, quienes pueden hacer referencia a un patrón bien conocido por su nombre en lugar de describir cada clase y los detalles del diseño. Esto acelera la aceptación o rechazo de una alternativa de diseño en particular.

Un patrón provee tres ventajas muy claras:

- La solución que aporta es bien conocida y por lo tanto ha sido muy bien testeada.
- Las ventajas y desventajas de su uso son conocidas anticipadamente a través de su documentación, y pueden ser tenidas en cuenta al momento del diseño y el desarrollo.
- Establecen un vocabulario común que facilita la comunicación entre "*stakeholders*".

Un patrón de diseño nombra, abstrae e identifica los aspectos claves de una estructura de diseño común. Esto lo hace útil para crear y reutilizar diseños orientados a objetos. Identifica las clases e instancias que participan, sus roles y colaboraciones y la distribución de responsabilidades. Cada patrón de diseño se enfoca en un problema o aspecto de diseño orientado a objetos en particular. Describe cuándo se aplica y dónde se puede aplicar, considerando otras restricciones de diseño junto a las consecuencias, ventajas y desventajas de su utilización.

Por otro lado, no existe un mejor patrón. Existen patrones específicos que aplican a problemas particulares. Algunos patrones son más útiles y comunes que otros. En el proceso de selección de un patrón, es aconsejable tener en cuenta que:

- La selección del patrón se debe apoyar sobre las similitudes existentes entre el contexto, el problema, y las fuerzas del patrón con un problema de diseño.
- Para resolver un problema en particular, podrían ser utilizados más de un patrón.

Todos los patrones están descriptos por una plantilla con secciones ya predefinidas. Este tipo de descripción permite una utilización sistemática, y ofrece una guía muy útil para el que tiene que escribir un patrón. En este trabajo se utiliza la plantilla definida en [Bus96], conocida como plantilla "*Pattern-Oriented Software Architecture*" (POSA).

3.5.3 Patrones de seguridad

En el área de conocimiento de la seguridad del software un reconocido principio de protección es la reutilización de recursos comunes, para evitar reinventar soluciones ad-hoc desde cero. Construir una solución desde cero (por ejemplo un protocolo de cifrado) es riesgoso debido a la probabilidad de fallos en el diseño; del mismo modo no es conveniente re-implementar desde cero una solución que es bien conocida, por los defectos que puedan introducirse en la etapa de desarrollo. En este aspecto los patrones de seguridad son una ayuda para hacer cumplir este principio de seguridad en la arquitectura y el diseño, ya que encapsulan el conocimiento experto en un formato reutilizable. Por otro lado, los patrones de seguridad deben incluir suficiente información detallada (incluso hasta el nivel de código) para ayudar a automatizar la fase de implementación. De este modo, los patrones de seguridad, se transforman en un medio adicional de proporcionar un nivel de seguridad a la construcción de software con requerimientos de seguridad.

Los principales objetivos que plantea la seguridad, son proteger la confidencialidad, integridad y disponibilidad de los datos. Los datos son un recurso muy valioso y se transforman con frecuencia en objetivo de ataques por parte de personas que desean ganar acceso a beneficios económicos, hacer un daño político o simplemente cometer vandalismo. Las contramedidas para garantizar los objetivos de seguridad se clasifican en cinco grupos [Fernandez 2007]: Identificación y Autenticación, Control de Acceso y Autorización, Logging, Criptogragía y Detección de Intrusión.

Los patrones de seguridad describen los mecanismos que caen dentro de estas categorías, o su combinación, destinados a detener o mitigar ataques, como así también los modelos abstractos que guían el diseño de estos mecanismos. Reúnen el extenso conocimiento acumulado sobre el aspecto seguridad con la estructura provista por los patrones de diseño. Juntos ofrecen lineamientos para la construcción y evaluación de sistemas seguros.

La seguridad como objeto de estudio, tiene una larga trayectoria, comenzando por los primeros modelos de Lampson [Lam71] y Bell/LaPadula [Fer06c] en los ’70 que resultaron en una variedad de aproximaciones para analizar los problemas de seguridad y diseñar mecanismos de seguridad. La evolución que ha tenido su tratamiento, influenciada indirectamente por las ideas de Christopher Alexander, al tratar de codificar esta experiencia en la forma de patrones de seguridad, en alguna medida se podría decir que ha sido natural. Era de esperar que ocurriera.

Yoder y Barcalow escribieron el primer trabajo sobre patrones de seguridad [Yod97]. Incluyeron una variedad de patrones útiles para resolver diferentes aspectos. Anterior a ellos, al menos tres trabajos [Fer93, Fer94, Ess97] mostraron modelos orientados a objetos de sistemas seguros, sin llamarlos patrones o sin utilizar una de las plantillas estándares para la representación de patrones. En 1998 se documentaron dos patrones más: un patrón para criptografía [Bra00] y un patrón para control de acceso [Das98]. Luego de esto, se presentaron algunos otros y hoy se pueden encontrar libros [Sch06, Ste05] y otros tipos de publicaciones. Hoy los patrones de seguridad están aceptados por muchas compañías tales como Microsoft [MSD 2011], Sun [JAV 2011] e IBM [IBM 2011], las que han hecho sus propias publicaciones y tienen sitios específicos en internet. También existe un sitio general sobre patrones de seguridad [sec 2011].

A pesar de todo ello, hay que destacar que la adopción generalizada de los patrones de seguridad está aún rezagada, especialmente cuando se compara con el éxito que han tenido los patrones de diseño de software. Cabría preguntarse en ese contexto:

- ¿Hay suficientes patrones de seguridad documentados?
- ¿Todos los patrones de seguridad son de utilidad para la construcción de soluciones de software?

- ¿Están correctamente documentados?
- ¿Son realmente útiles en la práctica?

Scandariato [Scandariato 2008] intenta responder estas preguntas en un trabajo donde presenta el estado del arte de los patrones de seguridad después de diez años de investigación y desarrollo, y las conclusiones a las que llega son las siguientes:

- La adopción limitada de los patrones de seguridad no es debida a una cobertura insuficiente del dominio de la seguridad; de todos modos la cobertura no es uniforme.
- Sólo la mitad de los patrones documentados y publicados son efectivamente activos, desde la perspectiva de la ingeniería de software. Es más, en este reducido conjunto hay un solapamiento significativo.
- La documentación existente, para los patrones de seguridad, necesita ser mejorada.
- Sólo en algunos casos, el patrón muestra ser una decisión realmente útil en la construcción de aplicaciones reales.

En este sentido, un aspecto relevante, para impulsar la utilización generalizada de los patrones de seguridad, es la construcción de métodos y herramientas que permitan estudiarlos y aplicarlos de manera sistemática en la mejora de la seguridad del software.

No es fácil precisar el significado, ni clasificar un patrón. Pero se hace necesario definir e identificar aquellos patrones de seguridad que forman el núcleo del conjunto de patrones conocidos: entendiendo como núcleo, aquellos patrones que pueden ser utilizados de manera constructiva cuando se intenta diseñar o definir la arquitectura de una aplicación. Al respecto, se han considerado las siguientes posibilidades para clasificar a los patrones de seguridad:

- **Un patrón de arquitectura:** Ya que es habitual que describa conceptos globales sobre la arquitectura de una aplicación.
- **Un patrón de diseño:** El hecho de que la seguridad puede ser considerada un aspecto de un sub-sistema de software ha llevado a ciertos grupos a considerar esta posibilidad.
- **Un patrón de análisis:** Las restricciones de seguridad deben ser definidas en el nivel más alto posible; a nivel de modelo conceptual de la aplicación. Por ejemplo, podemos definir los roles y los privilegios necesarios para que los usuarios puedan realizar sus tareas. Esto llevaría que al menos algunos patrones de seguridad son patrones de análisis.

Para ajustar el significado del término patrón de seguridad a los objetivos del trabajo, decimos que es "*…una relación entre cierto contexto, cierto conjunto de fuerzas o condiciones las cuales ocurren repetidamente en aquel contexto y ciertas configuración espacial que permite resolver o atender el conjunto de fuerzas o condiciones…*" [Cop 2011]. Esta definición menciona dos partes importantes: la descripción del problema (que incluye las fuerzas o condiciones) y la solución propuesta. A los fines prácticos de poder ser utilizado en el diseño de una aplicación, la descripción del problema debe estar enfocada y ser relevante para el ingeniero de software que intenta hacer uso del patrón. Los patrones que no cumplen con este requerimiento están fuera del foco de este trabajo. Un ejemplo es el patrón "*Nature of Safeguards*" [EH2003]. Sin una solución que pretenda resolver un problema propuesto, el patrón se transforma en una reescritura de los objetivos de seguridad. Este también es el caso de los patrones cuya expresión del problema es tan amplia como para plantearse una adecuada solución. Un ejemplo en este caso es el patrón "*Known Partners*" [SFBH+2006]. Esto tampoco implica que la mera presencia de la descripción de un problema a resolver junto a la solución propuesta, sea suficiente para que el patrón sea útil a los fines del ingeniero de software que pretende construir una aplicación segura. El nivel de abstracción y la factibilidad de construirse, son otros dos criterios que ayudan a distinguir entre patrones del núcleo y patrones que están fuera del núcleo.

Con respecto a la factibilidad de que el patrón de seguridad pueda ser construido por el ingeniero de software, Kienzle [KETEH02] observa lo siguiente:

> "*Dada la popularidad de los patrones de diseño en la comunidad de ingenieros de software, la inclinación natural es a asumir que cualquier cosa que lleve el nombre de patrón de seguridad debe ser descripto utilizando diagramas UML e incluir código fuente. Es verdad que muchos patrones de seguridad pueden ser presentados de este modo, pero hay otros patrones (referidos a procedimientos o a la arquitectura) que no ajustan en estas restricciones*".

El autor de este trabajo coincide con esta argumentación. Esto hace necesario identificar con claridad límites para encontrar aquellos patrones que sean factibles de utilizar con la metodología propuesta, la que apunta a construirlo como un artefacto de software y embeberlo dentro de los requerimientos de una aplicación segura.

Frecuentemente los patrones de seguridad describen el resultado de una solución (el qué), sin una estrategia para alcanzarla (el cómo). El patrón "*Role Based Access*" [KBZ01] sugiere crear roles conforme a responsabilidades y asignarles un conjunto apropiado de privilegios pero no suministra una guía para lograrlo. En muchos casos, esto se debe a que la solución no está implementada como un artefacto de software y depende de un diseñador que adhiera por ejemplo a un principio de menor privilegio o a una actividad a la cual deba estar subordinada el diseñador, como puede ser la identificación de activos o bienes a proteger en una etapa de evaluación de riesgos. Un ejemplo de esto es el patrón "*Roles*" [YB97] el cual sugiere utilizar los actores de los casos de uso como un punto de entrada cuando se defina el sistema de control basado en roles.

Por otro lado, muchos patrones son demasiado abstractos como para ser considerados parte del núcleo (no implementables por un ingeniero de software). Estos patrones, en general pueden ser subdivididos en dos categorías. Primero consideremos el caso del patrón "*Asset Valuation*" [SFBH+06]. Este patrón sugiere que sin una previa determinación de amenazas y vulnerabilidades, una organización no está en condiciones de evaluar adecuadamente los riesgos para sus bienes. De este modo, el valor económico e impacto sobre el negocio debe ser evaluado para cada bien. Esto representa una actividad que pertenece a la fase de análisis de riesgos, en lugar de presentar una solución implementable para la ingeniería de software. Segundo, considere el patrón "*Roles*" mencionado en el párrafo anterior. Está más próximo a ser una guía de mejores prácticas que un patrón del núcleo. Por el contrario, algunos patrones describen mecanismos de bajo nivel que son la implementación de técnicas o algoritmos y protocolos en lugar de soluciones basadas en patrones. Como ejemplo de esto, tenemos los protocolos de administración de claves de encriptación (*Session Key Exchange with Public Keys*) y las alternativas de implementación para gestionar datos de sesión (*Keep Session Data in the Client*). Un patrón debería describir la solución en términos de entidades de software construible y de pasos de diseño, o reglas para construir esas entidades. Por ejemplo, las entidades de software que constituyen la solución deberían estar correctamente dispuestas espacialmente y adecuadamente configurado su comportamiento. La solución debe contener entidades que se puedan mapear a elementos de software. En la tesis de intencionalidad/localidad de Eden y Kazman [EK03] se resume el criterio que permite distinguir entre las soluciones para una arquitectura, para un diseño o para una implementación, considerando el concepto de abstracción.

En conclusión, los patrones que forman el núcleo del conjunto, están definidos como patrones de seguridad con una descripción relevante del problema y una solución intencional, construible en términos de entidades de software, correctamente distribuidos espacialmente y con una configuración adecuada de su comportamiento que describe las iteraciones con participantes abstractos externos.

Se describen a continuación dos patrones de seguridad del núcleo. Primero el patrón de seguridad "*Packet Filter Firewall*" (PFFirewall) [Sch06] y segundo el patrón de seguridad "*Generic Object-Oriented Cryptographic Architecture*" GOOCA [Tropyc].

3.5.4 Descripción del patrón "Packet Filter Firewall"

3.5.4.1 Sinopsis

Filtra el tráfico de paquetes entrantes y salientes a una red, en base a inspección del nivel IP.

3.5.4.2 Contexto

Una red LAN conectada a Internet ó a otras redes con diferentes niveles de confianza. Un host de la LAN recibe y envía tráfico a Internet o a otras redes. Este tráfico se compone de varias capas o niveles. El nivel más básico es el nivel IP, formado por paquetes compuestos de encabezado y cuerpo. El cuerpo incluye la dirección de origen, la dirección de destino y otra información de ruteo, el cuerpo incluye el mensaje útil del paquete.

3.5.4.3 Problema

Las aplicaciones internas de una empresa son accedidas por un amplio espectro de usuarios que pueden intentar abusar de los recursos de la aplicación. Por otro lado, pueden ser numerosas y necesitar un control de acceso independiente y específico para cada una de ellas, lo cual puede hacer el sistema más complejo e inseguro.

3.5.4.4 Fuerzas (resumen de las consideraciones para aplicar el patrón)

Algunos de los hosts de otras redes pueden intentar atacar la red local utilizando el mensaje útil de un paquete IP. Este mensaje útil puede incluir virus o ataques a una aplicación específica. Es necesario identificar y bloquear aquellos hosts. La posible solución se ve limitada por las siguientes fuerzas:

- No es una opción aislar la red local ya que necesitamos comunicarnos con otras redes. De todos modos no se desea correr un alto riesgo por comunicarse con otras redes.
- Cualquier mecanismo de protección debe ser transparente para el usuario. El usuario no debería realizar acciones especiales para estar seguro.
- El costo directo e indirecto del mecanismo de protección debería ser relativamente bajo o el sistema será muy caro para implementar.
- Los ataques cambian constantemente; de modo que debería ser fácil cambiar la configuración del mecanismo de protección

3.5.4.5 Solución (describe una solución general al problema)

Un "*Packet Filter Firewall*" intercepta e inspecciona todo el tráfico de paquetes desde y hacia un puerto P (Figura 3.5.4.1) de un host de la red local. Aquellos paquetes que vienen desde, o van hacia, direcciones no confiables, son rechazados. Las direcciones no confiables se determinan a partir de un conjunto de reglas que implementan las políticas de seguridad de la institución. Un cliente de otra red sólo puede acceder al "*Local Host*" si existe una regla autorizando el tráfico desde esa dirección. Las reglas específicas pueden indicar una dirección o un rango de direcciones. Las reglas pueden ser positivias (permite el tráfico desde algunas direcciones) o negativas (bloquean el tráfico de algunas direcciones). La mayoría de los productos comerciales ordenan estas reglas para lograr un chequeo eficiente. Adicionalmente, si una petición de comunicación no se corresponde con alguna de las "Reglas Explícitas", entonces se aplica la "Regla por Defecto".

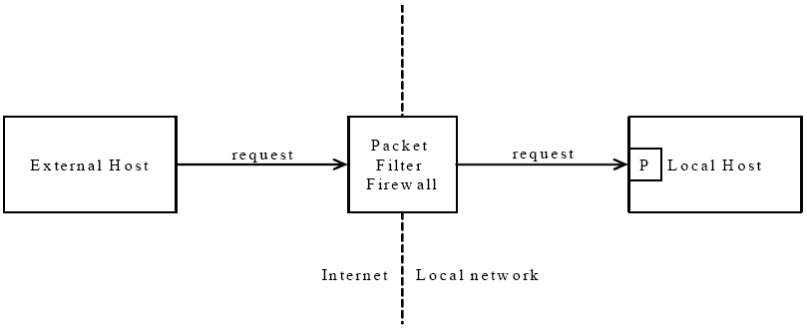

Figura 3.5.4.1. – Idea del Packet Filter Firewall

3.5.4.6 Estructura Estática: Diagrama de Clases

En la Figura 3.5.4.2 se muestra un "*External Host*" (Cliente) solicitando acceso a un "*Local Host*" (Servidor), a través de un "*Packet Filter Firewall*" (PFFirewall). Las políticas de la institución están embebidas en los objetos de la clase "*Rule*" y recogidas en la "*Rule Base*". "*Rule Base*" incluye estructuras de datos y operaciones para administrar las reglas de forma conveniente. En este conjunto las reglas están ordenadas y pueden ser "*Explicit*" o "*Default*".

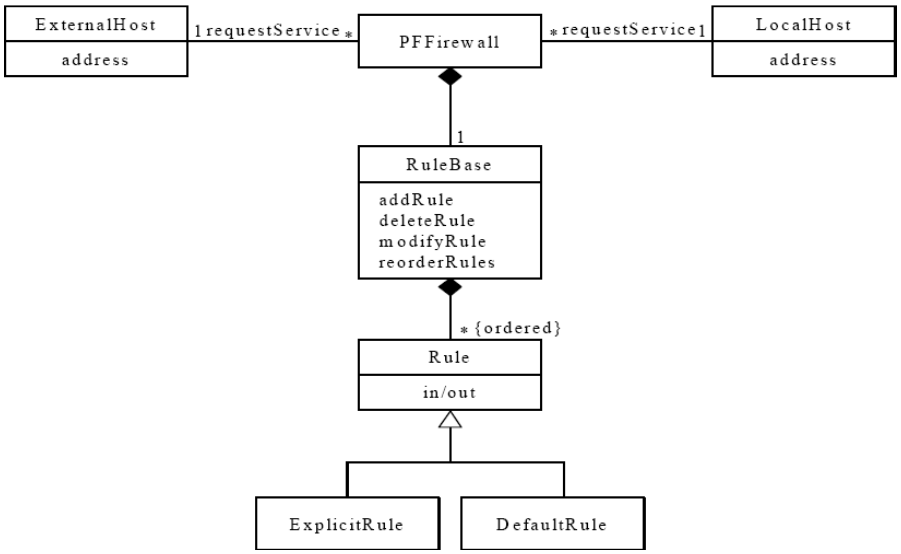

Figura 3.5.4.2 – Diagrama de clases del patrón PFFirewall.

3.5.4.7 Estructura Dinámica: Diagrama de Secuencia y Escenario

Se describen los aspectos dinámicos del "*Packet Filter Firewall*" usando un diagrama de secuencia para uno de los casos de uso básicos (Figura 3.5.4.3). Hay un caso de uso simétrico, Filtrando una petición saliente, el cual se omite por brevedad. Se omite también los casos de uso para sumar, remover o reordenar reglas los que son muy directos. Algunos autores describen los escenarios de casos de uso más informalmente.

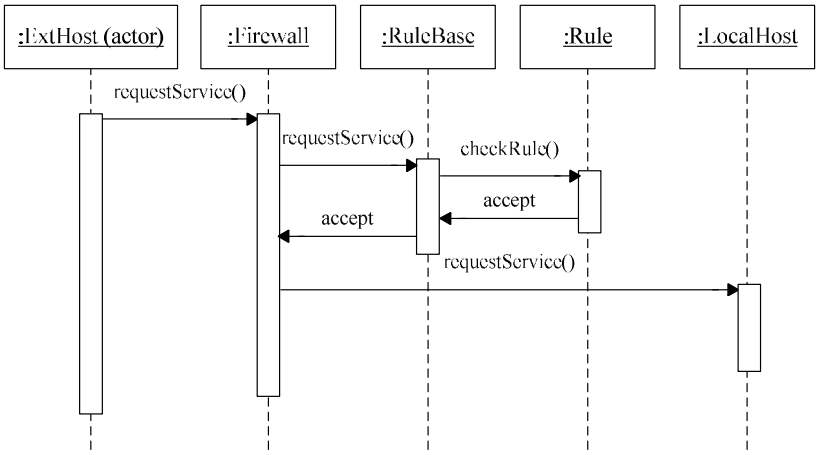

Figura 3.5.4.3 – Diagrama de secuencia para el filtrado de una petición de un Cliente

3.5.5 Descripción del Patrón "Generic Object-Oriented Cryptographic Architecture" (GOOCA)

3.5.5.1 Sinopsis

La criptografía moderna es ampliamente utilizada en muchas aplicaciones, tales como procesadores de texto, hojas de cálculo, base de datos y sistemas de comercio electrónico. El amplio uso de las técnicas criptográficas y el interés actual e investigación en arquitecturas de software y patrones llevó hacia las arquitecturas de software criptográfico y patrones criptográficos. Esta es una arquitectura criptográfica orientada a objetos genérica. Existe un conjunto de patrones criptográficos, basados en esta arquitectura, clasificados de acuerdo a los objetivos de la criptografía (confidencialidad, integridad, autenticación y no repudio).

3.5.5.2 Contexto

Dos objetos, Alice y Bob, intercambiando datos mediante mensajes. Necesitan realizar transformaciones criptográficas sobre los datos, solos o en combinación. Ellos desean disponer de un componente criptográfico que sea flexible y que pueda ser fácilmente reutilizable en otras transformaciones criptográficas.

3.5.5.3 Problema

Cómo diseñar una micro-arquitectura orientada a objetos para un diseño criptográfico y facilitar la reutilización de estos componentes…?

3.5.5.4 Aplicabilidad

- Cuando se deben cumplir aspectos que se presentan muy separados entre requerimientos criptográficos no funcionales y requerimientos funcionales de una aplicación específica.
- Cuando se debe lograr una transformación criptográfica exitosa y la aplicación a desarrollar debe ser independiente de aquella transformación.
- Cuando es necesaria una interface genérica para varios tipos de servicios criptográficos.

3.5.5.5 Fuerzas

- Minimizar la dependencia entre las funcionalidades criptográficas y el código de la aplicación, a fin de lograr los objetivos de reutilización.

- Incrementar la facilidad de lectura de programas con código criptográfico.
- Preservar el rendimiento de los algoritmos de transformaciones criptográficas.

3.5.5.6 Solución

Alice realiza una trasformación criptográfica sobre los datos antes de enviárselos a Bob. Bob recibe el mensaje y realiza la transformación inversa para recuperar los datos. Alice y Bob deben acordar la transformación a realizar y compartir o distribuir claves si fuese necesario.

3.5.5.7 Estructura Estática: Diagrama de Clases

En la Figura 3.5.5.1 se muestra el diagrama de clases que generaliza la transformación criptográfica en un interface abstracta y distingue los roles de Emisor y Receptor a partir de los roles de "*Codifier*" y "*Decodifier*". GOOCA es una abstracción de alto nivel para todos los patrones criptográficos. Todos los patrones criptográficos inician dicha estructura y su dinámica. GOOCA tiene dos clases, Alice y Bob, que representan el "todo" de una relación de agregación; y dos clases asociadas que representan la "parte". Ellas son Codifier y Decodifier. La clase Codifier tiene un método asociado f(), el cual realizar la transformación criptográfica sobre m. La clase Decodifier tiene su método asociado g(), el cual realiza la transformación criptográfica sobre x = f(m). La transformación y su reversa están basadas sobre el mismo algoritmo criptográfico.

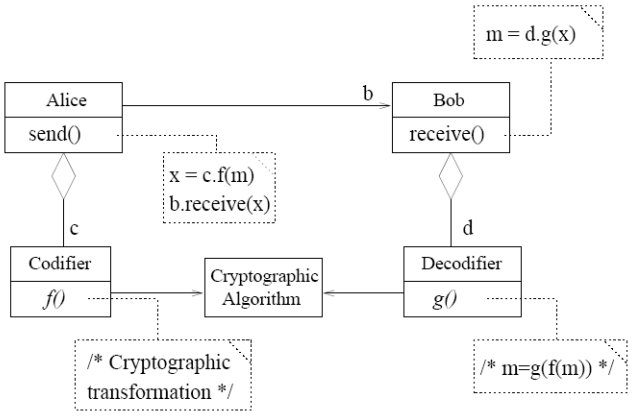

Figura 3.5.5.1 – Diagrama de Clases Patrón GOOCA

3.5.5.8 Estructura Dinámica: Diagrama de Secuencia

En la Figura 3.5.5.2 se muestra el comportamiento dinámico de GOOCA.

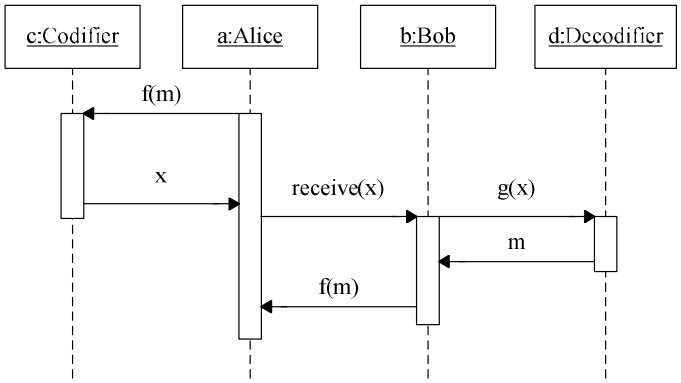

Figura 3.5.5.2 – Diagrama de Secuencia de GOOCA

3.5.5.9 Consecuencias

- Todas las transformaciones criptográficas presentarán un comportamiento común que podría ser generalizado en un diseño flexible.
- Las implementaciones específicas de instalaciones criptográficas podrían tener un mejor rendimiento que las instanciaciones de GOOCA, el cual está caracterizado por inserción de direcciones en el código.
- Se pueden obtener más fácilmente sistemas flexibles y adaptables, con requerimientos de seguridad basados en criptografía, cuando los algoritmos criptográficos son desacoplados de sus implementaciones, y a su vez éstas son desacopladas de los servicios criptográficos que utilizan.
- Los requerimientos de seguridad basados en criptografía son generalmente requerimientos no funcionales de aplicaciones de propósitos generales y no deberían contaminar la funcionalidad de aquellas aplicaciones. La separación explícita de aspectos en dos tipos de requerimientos facilita su lectura y reutilización.

3.5.5.10 Factores de implementación

- Este patrón puede ser fácilmente adaptado para tratar con el almacenamiento y recuperación de archivos. En esa situación, los mensajes del emisor y receptor pueden ser reemplazados por los de almacenamiento y recuperación. También se puede obviar la referencia a Alice y Bob.
- La "*Computer Reflection*" puede ser utilizada para hacer explícita la distinción de aspectos y restringir el código criptográfico a un meta-nivel, responsable de la intercepción de la comunicación o petición de almacenamiento y encriptación de los datos interceptados.
- Antes de que la comunicación segura comience, es necesario un paso de negociación previa "*handshake*" para que las dos partes acuerden sobre cuál transformación será realizada y para intercambiar información tal como claves y parámetros de algoritmos.
- El rol de Eve (tercero no confiable en el modelo) depende de la instanciación. Básicamente, puede ser reemplazar, modificar o insertar su propio mensaje dentro del canal de comunicación.

3.5.5.11 Ejemplo

Un sistema de pago electrónico, debido a sus fuertes requerimientos de seguridad, se diseña mejor como una instanciación de GOOCA. Esta aplicación, llamada PayPerClick, se diseña e implementa en el trabajo [Tropyc].

3.5.5.12 Usos conocidos

Todos los patrones criptográficos descriptos en la Figura 3.5.5.3., y ampliamente utilizados en sistemas como [HY97, Her97, CGHK98, HN98, BDR98] son instanciaciones de GOOCA.

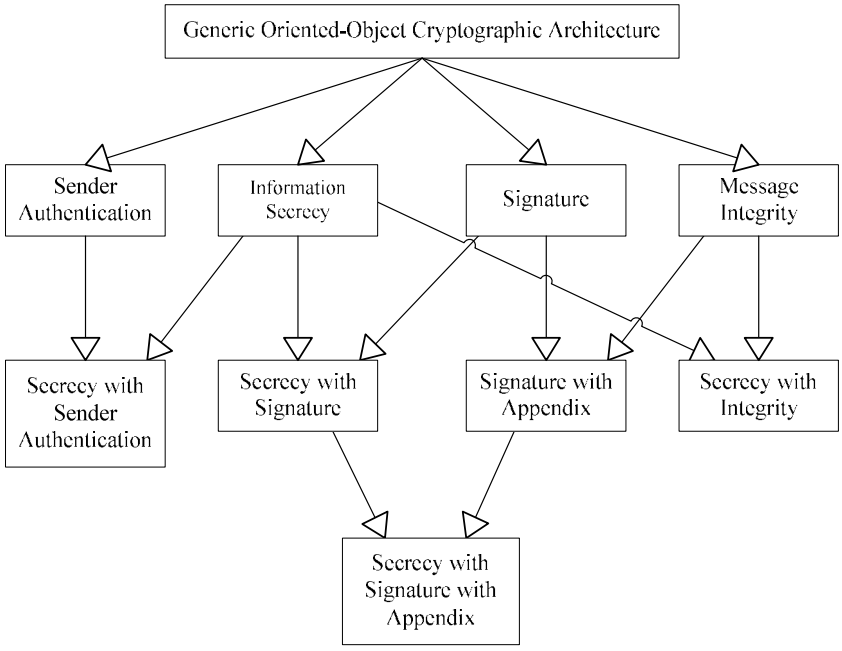

Figura 3.5.5.3. – Patrones de diseño criptográfico y sus relaciones.

3.5.5.13 Patrones relacionados

Al instanciar GOOCA, se pueden utilizar algunos patrones de diseño bien conocidos. El patrón "*Strategy*" puede ser utilizado para obtener independencia del algoritmo. El patrón "*Bridge*" puede ser utilizado para obtener independencia de la implementación. El patrón "*Abstract Factory*" puede ser utilizado en los pasos de negociación previa para decidir cuál algoritmo o implementación utilizar. Los patrones "*Observer*", "*Proxy*" y "*Client-Dispatcher-Receiver*" podrían ser combinados con los patrones criptográficos para ofrecer comunicaciones entre procesos de forma transparente y segura, de manera que Alice venga a ser parte del "*Forwarder*" y Bob ser incorporado dentro del "*Receiver*". El patrón "*State*" podría ser utilizado para proveer un comportamiento dependiente de los estados, tales como encender y apagar la seguridad del canal. El patrón "*NullObject*" puede ser utilizado para diseñar una transformación nula. El patrón "*Reflection*" puede ser utilizado para desacoplar la funcionalidad de la aplicación y la funcionalidad criptográfica.

3.6 Modelar Patrones de Seguridad con LEL y Escenarios

Para poder incorporar un Patrón de Seguridad al modelo contextual de una aplicación, construido con LEL y Escenarios, es imprescindible tener el Patrón de Seguridad modelado con los mismos elementos.

Es de esperar que la aplicación demande cumplir objetivos de seguridad ya sea porque se ha hecho un estudio de evaluación de amenazas previamente o porque se pueden identificar de manera directa requerimientos del dominio de la seguridad. En ambas situaciones, la solución propuesta consiste en poder implementar contramedidas que puedan ser implementadas utilizando Patrones de Seguridad.

Los Patrones de Seguridad construibles con entidades de software, tienen una descripción relevante del problema y una solución intencional, como propuesta de mitigar el problema. Poseen una relación entre un contexto, un conjunto de fuerzas o condiciones las cuales ocurren repetidamente en aquel contexto y una configuración espacial, modelada estática y dinámicamente, que permite resolver o atender el conjunto de fuerzas o condiciones. Esto es, una descripción del problema y una solución propuesta factible de ser construida mediante artefactos de software.

Pero los Patrones de Seguridad no ofrecen en la plantilla que los documenta un modelo contextual de los requerimientos que satisfacen, pero las experiencias realizadas para modelar aplicaciones con LEL y Escenarios, permitieron descubrir que también son artefactos adecuados para modelar, comprender y estudiar Patrones de Seguridad.

El proceso de construcción de un modelo contextual de requerimientos de un Patrón de Seguridad, como se puede ver en la Figura 3.6.1, se origina en la lectura y análisis de la documentación del Patrón. Al igual que en la construcción del modelo del dominio de una aplicación, se comienza por obtener información del Patrón de Seguridad. A partir de esta información se elabora una lista de símbolos que se deben conocer para entender su lenguaje: término del LEL. Se construyen los escenarios y se procede a deducir las CRC. Como se dispone en la documentación del Patrón de Seguridad de sus modelos estáticos y dinámicos, es posible validar las CRC encontradas. Luego, de existir alguna inconsistencia, se vuelve sobre la definición de LEL y Escenarios hasta obtener el conjunto de de CRC que modelan al Patrón de Seguridad. El resultado es un conjunto de términos de LEL y Escenarios que serán un modelo contextual del Patrón de Seguridad. Este modelo podrá integrarse con el correspondiente al dominio de la aplicación con requerimientos de seguridad que se pretende construir.

Esto permite construir una representación del Patrón de Seguridad que es posible incorporar en el estudio de una aplicación desde la etapa de elicitación, análisis y modelado de requerimientos. Luego es posible agregar a la plantilla de documentación del Patrón, un modelo contextual de los requerimientos que le dan origen, construido con LEL, Escenarios y CRC. Con el valor agregado de que las CRC han sido validadas en su cantidad y calidad con los modelos estáticos y dinámicos documentados en la plantilla del Patrón de Seguridad.

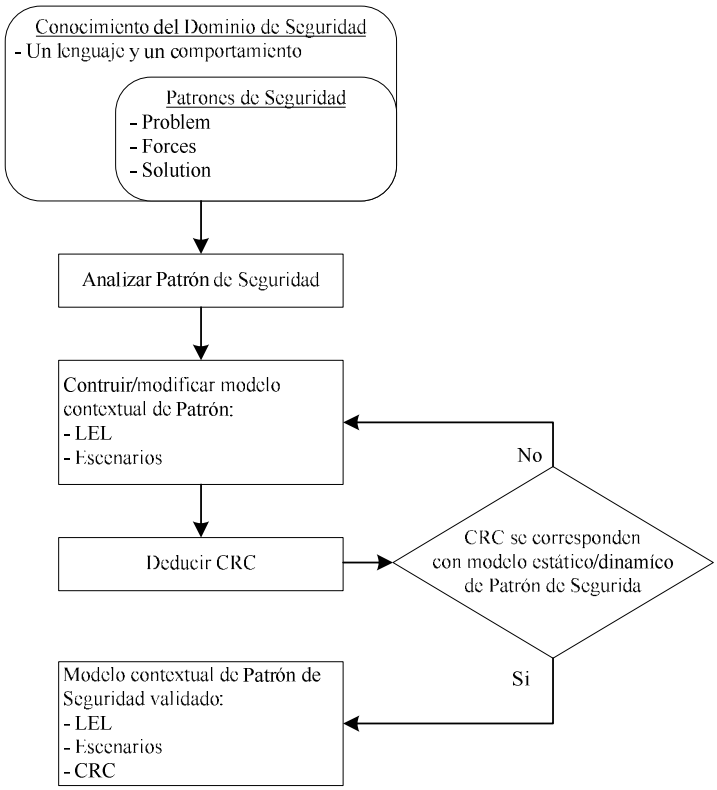

Figura 3.6.1 : Proceso de construcción de un modelo contextual de requerimientos para un Patrón de Seguridad

A continuación se modela con LEL y Escenarios, los requerimientos del Patrón de Seguridad GOOCA, presentado más arriba. Este patrón tiene la particularidad de poder ser utilizado para implementar más de un servicio de seguridad en la comunicación entre dos entidades abstractas representadas genéricamente por "*Alice*" y "*Bob*". Concretamente, puede ser utilizado para brindar servicios de confidencialidad, integridad, autenticación y firma electrónica.

3.7 LEL del Patrón de Seguridad GOOCA

A continuación se muestran las entradas de LEL encontradas para el patrón "GOOCA", producto del análisis de la documentación del patrón. Ellas son "*Alice" (*Tabla 3.7.1*); "Bob" (*Tabla 3.7.2*); "Codificador" (*Tabla 3.7.3*); Decodificador (*Tabla 3.7.4*); "Algoritmo de Encriptación" (*Tabla 3.7.5*).*

Sinónimo	Alice
Noción	Entidad que necesita mantener una comunicación confidencial con Bob
Impacto	Solicita al Codificador encriptar el mensaje con la Clave. Envía Mensaje encriptado a Bob. Conoce Clave.

Tabla 3.7.1. – Entrada de LEL "*Alice*"

Sinónimo	Bob
Noción	Entidad que necesita mantener una comunicación confidencial con Alice.
Impacto	Solicita al Decodificador desencriptar el mensaje encriptado con la Clave. Conoce Clave.

Tabla 3.7.2. – Entrada de LEL "*Bob*"

Sinónimo	Codificador
Noción	Entidad que encripta mensaje con una Clave.
Impacto	Conoce Algoritmo de Encriptación.

Tabla 3.7.3. – Entrada de LEL "Codificador"

Sinónimo	Decodificador
Noción	Entidad que desencripta un mensaje encriptado con una Clave
Impacto	Conoce Algoritmo de Encriptación.

Tabla 3.7.4. – Entrada de LEL "Decodificador"

Sinónimo	Algoritmo de Encriptación
Noción	Transformación matemática para Encriptar y Desencriptar un mensaje con una Clave.
Impacto	Es utilizado por Codificador y Decodificador.

Tabla 3.7.5. – Entrada de LEL "Algoritmo de Encriptación"

3.8 Escenario del Patrón de Seguridad GOOCA

Se describe el escenario identificado en el dominio del patrón GOOCA en la Tabla 3.8.1. Este escenario modela la dinámica de un intercambio de información entre Alice y Bob con un servicio de confidencialidad. Lo cual no le quita generalidad ya que puede ser extendido para brindar servicios de integridad y autenticación.

Título	Envía Mensaje encriptado con Clave
Objetivo	Ocultar el contenido del Mensaje
Contexto	Comunicación establecida entre Alice y Bob.
Recursos	Mensaje, Clave
Actores	Alice, Bob
Episodios	Alice envía Mensaje y Clave a Codificador. Codificador encripta Mensaje utilizando Algoritmo de Encriptación y Clave. Codificador devuelve Mensaje encriptado a Alice. Alice envía Mensaje encriptado a Bob. Bob envía Mensaje encriptado y Clave a Decodificador. Decodificador desencripta Mensaje encriptado utilizando Algoritmo de Encriptación y Clave. Decodificador devuelve Mensaje a Bob. Bob lee Mensaje.

Tabla 3.8.1. – Escenario "Envía Mensaje encriptado con Clave"

3.9 CRC Deducidas

A partir del LEL y el Escenario propuesto para el patrón de seguridad y utilizando el algoritmo de derivación de tarjetas CRC implementado en BMW, se obtienen las siguientes CRC primarias: "*Alice*" (Tabla 3.9.1), "*Bob*" (Tabla 3.9.2) y secundarias: "Codificador" (Tabla 3.9.2), y "Decodificador" (Tabla 3.9.4). Luego se muestra nuevamente el modelo estático del patrón GOOCA, donde se pueden identificar claramente las CRC deducidas como clases de diseño en el modelo estático del patrón. Ver Figura 3.9.1.

Primary CRC Card	*Alice*
Responsabilities	Solicita al Codificador encriptar el mensaje con la Clave. Envía Mensaje encriptado a Bob. Conoce Clave.
Collaborations	*BOB, CODIFICADOR, DECODIFICADOR*

Tabla 3.9.1. – CRC primaria "Alice"

Primary CRC Card	*Bob*
Responsabilities	Solicita al Decodificador desencriptar el mensaje encriptado con la Clave. Conoce Clave.
Collaborations	*CODIFICADOR, DECODIFICADOR, ALICE*

Tabla 3.9.2. – CRC primaria "Bob"

Secondary CRC Card	*Codificador*
Responsabilities	Conoce Algoritmo_de_Encriptación.
Collaborations	*ALICE, BOB, DECODIFICADOR*

Tabla 3.9.3. – CRC secundaria "Codificador"

Secondary CRC Card	*Decodificador*
Responsabilities	Conoce Algoritmo_de_Encriptación.
Collaborations	*BOB, ALICE, CODIFICADOR*

Tabla 3.9.4. – CRC secundaria "Decodificador"

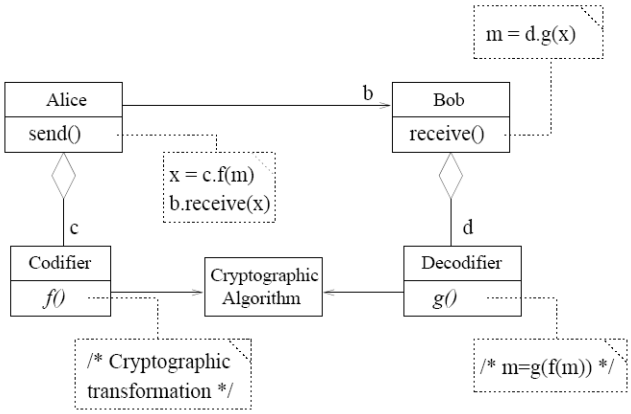

Figura 3.9.1 – Diagrama de Clases de diseño Patrón GOOCA

3.10 Conclusión del capítulo

Es posible construir modelos contextuales de Patrones de Seguridad utilizando LEL y Escenarios tomando como fuente para la elicitación de requerimientos, la plantilla del Patrón. Puntualmente el problema, las fuerzas y la solución propuesta. Así también lo muestra el trabajo publicado en CACIC 2010, [Solinas 2010], donde es posible verificar el uso de otros Patrones de Seguridad en la construcción de sistemas seguros para la Ingeniería en Computación.

Por otro lado, las experiencias realizadas para modelar Patrones de Seguridad con LEL y Escenarios han sido puntuales, sobre un conjunto reducido. Pero si se tiene en cuenta que existe un "*core*" de Patrones de Seguridad [Scandariato 2008] que describen una solución en términos de entidades de software construibles, correctamente dispuestas espacialmente y con adecuados comportamientos configurados en los modelos estáticos, es perfectamente factible esperar que estos patrones puedan ser representados con un modelo contextual de requerimientos, que en definitiva busca mapear artefactos deducidos a partir del mismo, con las entidades abstractas de los modelos estáticos documentados en el Patrón de Seguridad.

4 Elicitación de requerimientos utilizando patrones de seguridad

En el Capítulo anterior se mostró que es posible construir un modelo contextual que capture el lenguaje y el comportamiento de un Patrón de Seguridad utilizando LEL y Escenarios. Deducir a partir de ellos tarjetas CRC utilizando la heurística implementada en el BMW y finalmente validar dichas tarjetas con los artefactos que componen el modelo estático documentado en la plantilla del Patrón de Seguridad. Resumiendo, los pasos fueron los siguientes:

i) Analizar información referida al Patrón de Seguridad.

ii) Construir un modelo contextual.

iii) Deducir CRC.

iv) Validar CRC obtenidas con la documentación del Patrón de Seguridad.

Entonces, tenemos posibilidad de utilizar esta nueva representación de un Patrón de Seguridad en la construcción de un modelo contextual de requerimientos de una aplicación con objetivos de seguridad. En esta sección se describe un método que nos permite lograr esto de forma sistemática y pueda luego trasladarse este conocimiento a la construcción de nuevas aplicaciones.

4.1 Objetivo

La meta que persigue este Capítulo es describir un método de construcción de software seguro que aplique el conocimiento adquirido con LEL y Escenarios, a una aplicación con objetivos de seguridad y utilizar Patrones de Seguridad como las mejores prácticas para satisfacer los objetivos de seguridad.

El método de construcción de software seguro, utiliza LEL y Escenarios y la estrategia que de forma sistemática propaga los cambios sobre las CRC [Antonelli 2003]. Sobre esta base, se incorporan Patrones de Seguridad. Los cambios que se produzcan sobre los términos de LEL y Escenarios que modelan la aplicación y el Patrón de Seguridad, deben impactar en el conjunto de CRC que representan el modelo de análisis de la aplicación con objetivos de seguridad. Una síntesis de la metodología se puede ver en la Figura 4.1.1.

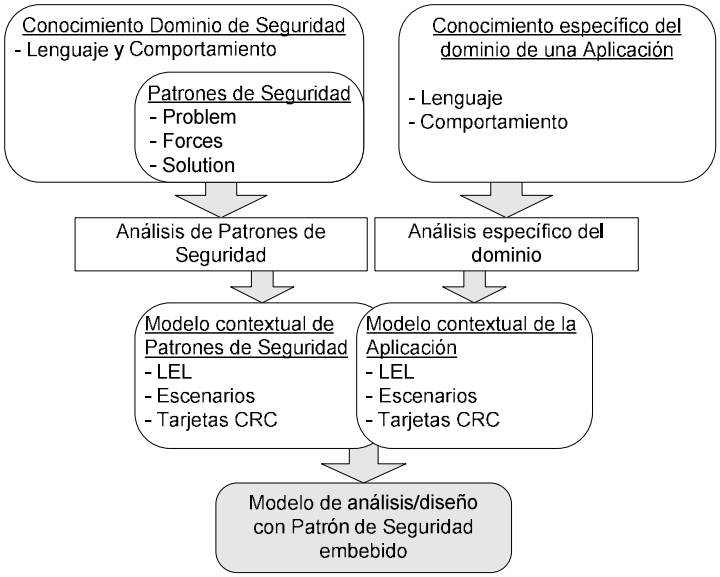

Figura 4.1.1. – Descripción de Metodología

En la Sección 4.2 se presenta una síntesis del método aplicado luego en construcción de una aplicación con requerimientos de seguridad que se describe en la Sección 4.3. En la Sección 4.4 se hace un modelo de amenazas para la aplicación. En la Sección 4.5 se construye un LEL para la aplicación y un conjunto de escenarios a partir de los casos de uso de los requerimientos funcionales. Luego se deducen las CRC que permiten construir una primera aproximación de un modelo estático de clases candidatas. Como la aplicación tiene requerimientos de seguridad, se los identifica y se los asocia a un Patrón de Seguridad, cuyo modelo contextual de requerimientos con LEL y Escenarios ha sido construido en el Capítulo anterior. Se integran ambos modelos y se presenta una solución definitiva para la aplicación en forma de modelo estático.

4.2 Síntesis del método utilizado y sus etapas

Hay decisiones que son determinantes al momento de necesitar producir software seguro. La decisión sobre el elección del método para elicitar y analizar requerimientos es una de ellas. En cada etapa del proceso de desarrollo existen principios de seguridad que deben aplicarse si se pretende lograr un producto seguro. Y hasta sería deseable [Fernandez 2007_1] que en cada etapa pueda testearse la conformidad con esos principios utilizando algún tipo de métrica que permita evaluar el potencial error en la interpretación de su aplicación. Si no hay evidencia de que se ha realizado una tarea en la dirección de las mejores prácticas, para lograr un software seguro, no habrá motivo para considerar que se han minimizado sus vulnerabilidades.

Asumiendo que la aplicación a desarrollar requiere servicios de seguridad. Ya sea porque se ha aplicado un modelo para evaluación de amenazas y se identificaron bienes a proteger. Ó que es necesario dar cumplimiento a políticas de seguridad. O quizás existen requerimientos que ponen en evidencia cumplir con objetivos de seguridad. En todos los casos, para construir la aplicación, es necesario un ciclo de desarrollo de software seguro como el propuesto por Fernandez [Fernandez 2006] el cual se esquematiza en la Figura 4.2.1., donde se han incorporado los artefactos utilizados en este trabajo: LEL, Escenarios, CRC y Patrones de Seguridad. Estas etapas si bien generales, pueden ser interpretadas como partes integrantes de un proceso lineal o un proceso evolutivo. Las flechas blancas indican los artefactos a utilizar en la etapa para aplicar la seguridad y las flechas negras indican dónde debe chequearse el cumplimiento de los principios de seguridad.

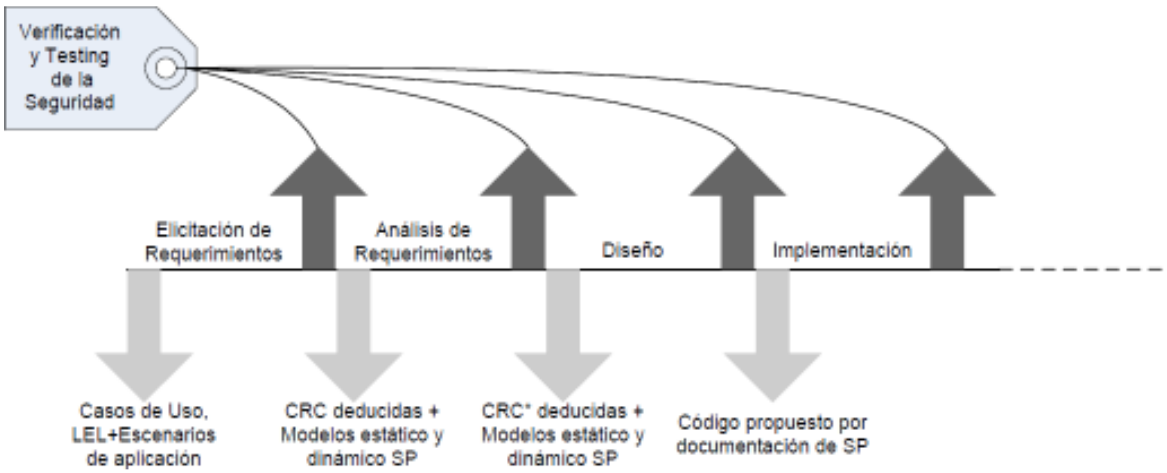

Figura 4.2.1. – Ciclo de desarrollo de software seguro.

Entre los múltiples trabajos que destacan la importancia de los requerimientos para la producción de software de calidad y por extensión, software seguro, me permito mencionar a Mizuno, quien organiza los errores que se pueden producir a lo largo del proceso de desarrollo de software, en un modelo de catarata de errores [Mizuno 1983]. El modelo muestra que una especificación incorrecta de requerimientos puede ocasionar errores ocultos dentro del sistema (Figura 4.2.2). Si trasladamos este modelo a una aplicación con requerimientos de seguridad, el resultado se agudiza, desde el momento en que se sabe que dicha aplicación requiere cumplir objetivos de seguridad. Es entendible que una incorrecta especificación de requerimientos de seguridad y un diseño incorrecto de los mismos se traduzca en aplicaciones más vulnerables por desconocimiento ya no de errores, sino de amenazas ocultas, amenazas no identificadas o amenazas identificadas que no son tratados adecuadamente por falta de interés. Situación a la que se ha llegado por un inadecuado tratamiento de la seguridad desde las etapas tempranas del proceso de desarrollo de software.

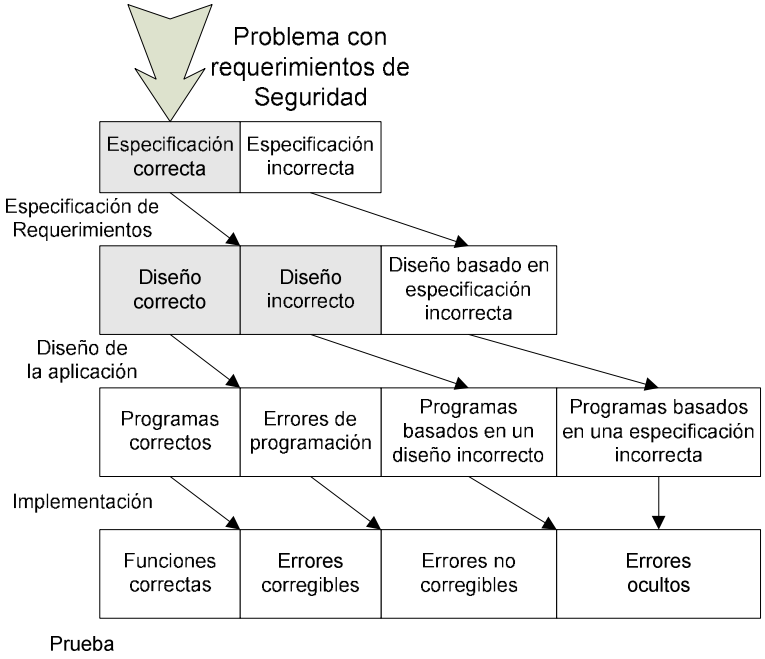

Figura 4.2.2. – Catarata de errores de Mizuno.

Por ello, cualquier método que pretenda atacar la construcción de software seguro, debe presentar una propuesta que enfoque el problema desde las etapas más tempranas del proceso de desarrollo: elicitación y análisis de requerimientos y diseño. El principio aceptado es incorporar las mejores prácticas en forma de Patrones de Seguridad como medio de garantizar la seguridad desde estas etapas.

Otros autores han trabajado intentando aplicar el mismo principio [He&Anton, 2004; Konrad, Cheng, Campbell, &Wassermann, 2003; Zuccato, 2004] y han desarrollado soluciones para poder integrar los Patrones de Seguridad pero siempre en etapas más próximas a la de diseño. Sumando directamente las soluciones propuestas por los modelos estáticas (diagramas de clases) a los diagramas de objetos de la realidad de los sistemas de software a construir.

Por otro lado, en la elicitación de requerimientos de una aplicación, encontramos un conocimiento propio del dominio, al que pertenece la aplicación. Y hay un conocimiento propio del dominio de la seguridad que ha evolucionado, ha sido estudiado y está documentado en términos de Patrones de Seguridad. Representan las mejores prácticas en el desarrollo de software seguro. Cada

uno de estos dominios tiene su propio lenguaje, su propia semántica, su propio comportamiento, sus propios especialistas. Deberían poder integrarse naturalmente y permitir la construcción de modelos de requerimientos, análisis y diseño que tengan en cuenta la seguridad presentada en forma de Patrones de Seguridad, como se muestra en la Figura 4.2.3.

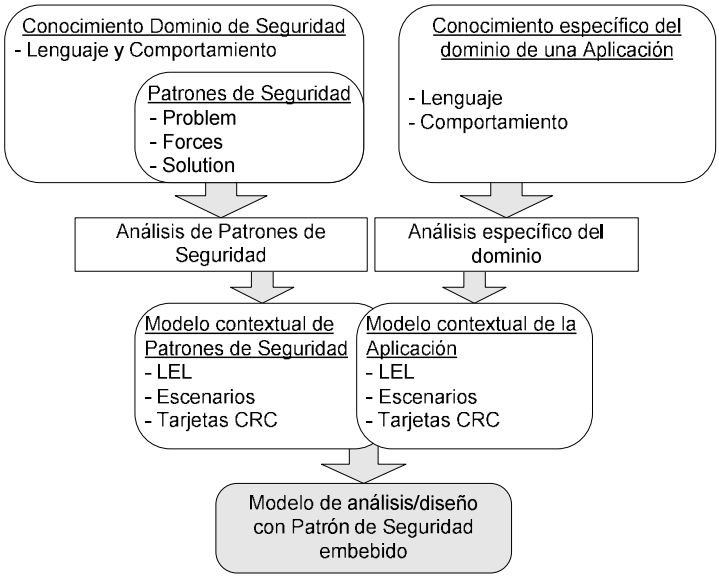

Figura 4.2.3. – Descripción de Metodología

Los Patrones de Seguridad así modelados, integrados al modelo de requerimientos, análisis y diseño de la aplicación, cumplen una función relevante por ser artefactos completamente definidos y documentados. En contraste con la aplicación que se pretende construir que debe definirse y documentarse. Teniendo en cuenta estos dos aspectos, completitud y documentación, es que los Patrones de Seguridad seleccionados para construir la aplicación, cumplen el rol de conducir las decisiones de diseño de la aplicación con requerimientos de seguridad. No son elementos asépticos y es de esperar que así lo sea. Por un lado tenemos un artefacto que está definido completamente, está documentado, definido hasta la etapa de diseño e incluso en algunos casos hasta la de implementación. Por el otro lado tenemos una aplicación que es todo potencialidad, es únicamente requerimientos. Si pensamos que las decisiones que se toman al modelar los requerimientos de la aplicación, pretenden avanzar hacia las etapas de análisis y diseño, el Patrón de Seguridad cumple la función de facilitar y ayudar con ese avance. El Patrón de Seguridad seleccionado conduce las decisiones hacia la etapa de implementación y ya embebido en la aplicación, cumple con un segundo rol no menos relevante. Al estar completamente definido y documentado, se transforma en un elemento con el cual comparar el diseño propuesto de la aplicación. En esos términos ayuda a verificar su correcta implementación.

Según [Yoshioka 2008], desde el punto de vista del ciclo de vida de un patrón de diseño, las actividades que tienen que ver con el uso de los mismos, pueden ser clasificadas en dos grupos: extracción e implementación. El proceso de extracción da como resultado un catálogo de patrones documentados. Mientras que el proceso de aplicación está compuesto de varias actividades: A1) Reconocimiento del contexto y problema en el desarrollo de software; A2) Selección del patrón de software adecuado que resuelve el problema; A3) Aplicar el patrón seleccionado y A4) Evaluar el resultado de su aplicación.

Con la ayuda de estas consideraciones, es posible delinear un método que las tenga en cuenta y permita la construcción de una aplicación con requerimientos de seguridad incorporando naturalmente Patrones de Seguridad para producir software seguro. A continuación, se describen sus etapas.

4.2.1 E1: Modelar Patrones de Seguridad.

El primer paso es construir modelos contextuales de Patrones de Seguridad utilizando LEL y Escenarios tomando como fuente para la elicitación de requerimientos, la plantilla del Patrón; específicamente el problema, las fuerzas y la solución propuesta. Esto contribuye a la actividad A1 de Yoshioka, al asociar al Patrón de Seguridad con la misma semántica que tendrá la descripción del contexto de la aplicación que se desea construir.

Luego, utilizando la nueva versión del BMW, es posible deducir el conjunto de CRC para cada patrón. Y validar el conjunto de CRC así obtenidas, en cantidad y calidad, con los modelos estáticos y dinámicos documentados en el Patrón de Seguridad. Es un modo de garantizar que el modelo construido con LEL, Escenarios y CRC se ajusta a los modelos estático y dinámico propuestos en la documentación. Esto transforma a las CRC de análisis (deducidas por la heurística implementada en el BMW) con sus relaciones y responsabilidades, en CRC de diseño, ya que tienen el respaldo de clases identificadas en los modelos estático y dinámico de la documentación del Patrón de Seguridad.

Este juego de ida y vuelta, ayuda a involucionar el Patrón de Seguridad, en la búsqueda de términos de LEL y Escenarios que lo representen. Es un aporte novedoso y se transforma en la única herramienta que tiene a su alcance, quien modele requerimientos con LEL y Escenarios y pretenda construir una aplicación con requerimientos de seguridad utilizando Patrones de Seguridad.

Cada Patrón de Seguridad así modelado queda identificado con un conjunto de términos de LEL que modelan su propio lenguaje y uno o más Escenarios que modelan su comportamiento dinámico. Ambos expresados en lenguaje natural y en una semántica que permite identificar el Escenario con la funcionalidad del Patrón de Seguridad.

4.2.2 E2: Modelar Aplicación.

Se debe construir un modelo contextual de la aplicación partiendo de la elaboración de un LEL. Luego, determinados los Casos de Uso, elicitando los requerimientos funcionales y con el LEL, se construyen los Escenarios.

En el proceso de construcción de Escenarios se aplica el principio de circularidad y vocabulario mínimo, ya utilizado en la construcción del LEL. Posteriormente se deduce el conjunto de CRC de análisis de la aplicación, utilizando la heurística implementada en el BMW.

Al finalizar esta etapa se dispone de los siguientes elementos:

- Un conjunto de términos de LEL que modelan el lenguaje propio del dominio de la aplicación.
- Un conjunto de Escenarios con el comportamiento dinámico de los requerimientos funcionales de la aplicación.

4.2.3 E3: Identificación de Patrones de Seguridad.

Con los elementos obtenidos en las etapas anteriores, se continúa con la identificación de Patrones de Seguridad, lo cual está contribuyendo a la actividad A2 propuesta por Yoshioka. En esta instancia pueden ocurrir dos cosas:

- Se identificaron claramente objetivos de seguridad, en la construcción de los Casos de Uso, que deberán ser mapeados sobre Patrones de Seguridad. Lo cual es coherente con la metodología que propone la utilización de casos de mal uso o “misuse case”.
- Se descubrieron objetivos de seguridad en la descripción de los episodios de los Escenarios de la aplicación, que también serán mapeados sobre Patrones de Seguridad. Esto es como resultado de aplicar el principio de circularidad y vocabulario mínimo extendido a la construcción de Escenarios, donde un Patrón de Seguridad queda

identificado con el Nombre de un Escenario y una semántica identificada con su funcionalidad.

En ambos casos, dado que todos los artefactos están expresados en lenguaje natural, la semántica propia de los artefactos hace posible su identificación y mapeo. Desde ya que disponer de la plantilla completa del Patrón de Seguridad para su consulta, es un elemento a tener en cuenta en esta instancia.

Como resultado de esta etapa se tendrá identificado el/los Patrones de Seguridad que deberán conducir la construcción de un modelo que los integre en la Aplicación. Esta integración se dará a partir de la identificación de las CRC de la Aplicación que tienen sus homólogas entre las CRC del Patrón de Seguridad o que comparten sus responsabilidades. Es decir, habrá CRC de la Aplicación que se podrán asociar con CRC del Patrón de Seguridad (asociadas con clases de diseño) a partir de las funciones comunes que realizan en contextos diferentes.

4.2.4 E4: Análisis de CRC Primarias.

Identificados los Patrones de Seguridad, conociendo cuáles serán las mejores prácticas para cumplir con los objetivos de seguridad identificados, se procede a analizar las CRC primarias encontradas para la aplicación. El objetivo es identificar coincidencias entre sus responsabilidades y las responsabilidades de las CRC primarias encontradas para el Patrón de Seguridad. Se observa que si el Patrón de Seguridad es correctamente seleccionado, aparecen responsabilidades comunes.

Para mantener la consistencia y trazabilidad de requerimientos, en el modelo construido con BMW, no es posible modificar directamente las CRC. Las CRC Primarias son el resultado de aplicar una heurística a los términos de LEL que aparecen como actores en la descripción de los Escenarios.

Para lograr un conjunto de CRC primarias homólogas a las del Patrón de Seguridad, hay que redefinir los impactos de los términos de LEL que dieron origen a las CRC Primarias de la aplicación.

Como resultado tendremos un nuevo conjunto de términos de LEL que darán lugar a las nuevas CRC Primarias validadas con los modelos estáticos de diagramas de clase del Patrón de Seguridad. Esta tarea se debe hacer teniendo siempre en vista, de manera directa, las CRC del Patrón de Seguridad. Y de manera indirecta, sus modelos estáticos y dinámicos documentados.

4.2.5 E5: Analizar las CRC secundarias.

Las CRC Secundarias del Patrón de Seguridad, son aquellas que colaboran con las CRC Primarias y no aparecerán entre las CRC Secundarias obtenidas para la Aplicación. Se trata de términos de LEL que no han sido mencionados entre los impactos de los términos de LEL que dieron lugar a las CRC Primarias de la Aplicación. En un principio no deberían modificarse sustancialmente los términos de LEL que dan origen a las CRC Secundarias del Patrón de Seguridad. De todos modos es necesario revisarlos a fin de mantener la consistencia del modelo.

4.2.6 E6: Analizar Escenarios.

Está muy ligado al 3er.Paso, ya que si el Patrón de Seguridad se identificó directamente como un Caso de Uso es necesario modelarlo en términos del contexto. En esta situación directamente se utilizan los términos de LEL vinculados al Patrón de Seguridad en el contexto de la aplicación que se desea construir.

Para el caso que se hayan identificado los patrones en los episodios de los escenarios, hay que reescribir los episodios de estos Escenarios a fin de mantener la consistencia del modelo, aplicando la

regla de circularidad y vocabulario mínimo, ahora a los Escenarios. Luego agregar al modelo el Patrón de Seguridad identificado.

4.2.7 E7: Deducción de CRC.

Ya se tienen todos los elementos para proceder naturalmente con la herramienta BMW y deducir el conjunto de CRC que permitirán completar el diseño de la Aplicación. Aquí claramente estamos aplicando el Patrón de Seguridad identificado, lo que contribuye a la actividad A3 propuesta por Yoshioka. Tener en cuenta, en todo momento, que a partir de este conjunto de CRC se puede construir el nuevo diagrama de objetos de la realidad que ahora contiene abstracciones que son de diseño. Ellas han sido propuestas por los modelos estáticos y dinámicos del Patrón de Seguridad. De este modo, si se decide utilizar el Patrón de Seguridad en la construcción de la aplicación, no deberían tomarse decisiones que concluyan en una inadecuada implementación del Patrón de Seguridad, sugerida por sus modelos: estático y dinámico. Al finalizar esta etapa se debería tener un modelo estático de objetos de la realidad/clases de diseño y un modelo dinámico que permita verificar la correcta utilización de los elementos del Patrón de Seguridad.

4.2.8 E8: Reutilización de Código.

Por último, si el Patrón de Seguridad tuviese código con su implementación, y ese código coincidiera con el que se utilizará en la implementación, su utilización es directa y el método permitiría llegar hasta la etapa de implementación. Esto permitiría validar una correcta aplicación del Patrón de Seguridad en la etapa de implementación, lo que estaría de acuerdo con la actividad A4 propuesta por Yoshioka.

De este modo, el método soporta completamente el proceso de aplicación de Patrones de Seguridad propuesto por Yoshioka.

4.3 Descripción de la aplicación utilizada

Sea una terminal de auto-consulta con dispositivo "*Encrypted Pin Pad*" (EPP) similar al que disponen los cajeros automáticos y cuya funcionalidad se describe en "Anexo I: Descripción de EPP". Este tipo de terminal permite que los clientes de un banco, por ejemplo, puedan consultar el estado de sus productos o transacciones. Para lograrlo, la terminal demanda construir un "*driver*" para el dispositivo EPP. Luego, mediante una Aplicación Cliente Genérica (ACG) se podrán ejecutar, a modo de "*testing*", todas las funcionalidades del EPP. La ACG será reutilizada por las entidades bancarias en el desarrollo de sus aplicaciones de auto-consulta específicas.

El diseño y construcción completa de esta aplicación corresponde al Proyecto Integrador de la carrera de Ingeniería en Computación "*Construcción de Driver de usuario para dispositivo Encrypted PIN Pad utilizando Desarrollo Conducido por Modelos*", del Ing. Gastón Nicolás Medina, que asesoré en calidad de Director en el año 2009/2010.

Para construir el modelo de comunicación entre la ACG y dispositivo EPP, se propone la implementación de un "*middleware*" que permita a la ACG realizar una abstracción del "*hardware*" y a la vez proporcionar una interface para interactuar de forma segura y amigable. Se opta por un "*middleware*" del tipo "*Remote Procedure Calls*" (RPC) que se sustanciará como un Driver o controlador. En la Figura 4.3.1 se representa el modelo de comunicación propuesto.

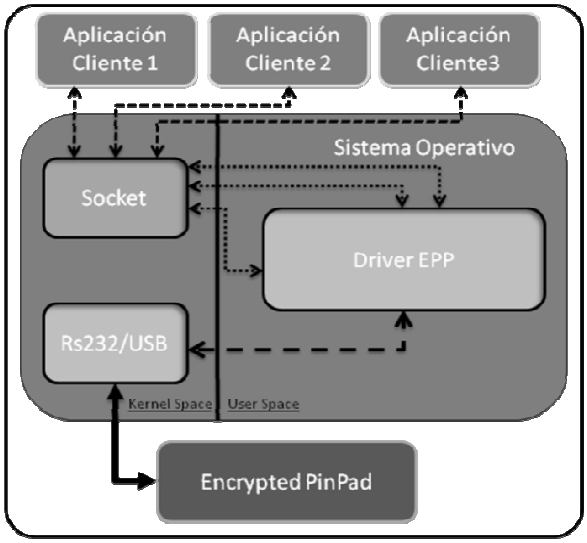

Figura 4.3.1. – Arquitectura de Comunicación

El Driver a diseñar y construir para el dispositivo EPP debe proveer una interface entre una aplicación de alto nivel, "Aplicación Cliente n" a desarrollar por el cliente final de este producto, y el modulo del "*kernel*" del Sistema Operativo que administra el enlace físico con el dispositivo EPP. Debido a la naturaleza del entorno donde estará incluida la aplicación de alto nivel, el Driver no debe, bajo ninguna situación, bloquear o inhabilitar el sistema, permitiendo además la posibilidad de recargar o restaurar su ejecución sin reiniciar el sistema. Estos requerimientos llevan a que el Driver deba implementarse como un Driver en Modo Usuario.

El Sistema Operativo para el que se debe desarrollar el Driver es Windows XP, por tanto deberá comportarse como un Servicio del Sistema Operativo ejecutándose como un proceso residente en "*background*".

El Driver le debe permitir a la ACG, ejecutar funciones específicas (Tabla 4.3.1) y generales (Tabla 4.3.2) del EPP. Deben poder ejecutarse de forma independiente a fin de permitir corroborar, parte por parte, que una función compuesta sea validada en forma parcial y total. Estas funciones luego serán utilizadas por la aplicación de auto-consulta que desarrolle el cliente.

Función	Descripción
Ingresar PIN	Habilitar el ingreso de una serie de dígitos que conformen un PIN (ISO9564-1; formato 0) controlados por un tiempo de expiración. El valor del PIN ingresado debe ser enviado a la aplicación Cliente.
Ingresar Dígitos	Habilitar el ingreso de dígitos individuales, controlados por un tiempo de expiración. El valor del dígito ingresado debe ser enviado a la aplicación Cliente.
Ingresar Claves de Administradores	Habilitar el ingreso de una serie de dígitos que conformen la clave del primer y segundo administrador, controlado por un tiempo de expiración. El EPP informa la finalización de la operación a la aplicación Cliente. Ambas claves de administrador se ingresan y se salvan en el EPP.
Modificar Claves de Administradores	Habilitar el ingreso de una serie de dígitos que conformen la nueva primera o segunda clave de un administrador, controlado por un tiempo de expiración. El EPP informa la finalización de la operación a la aplicación Cliente. La nueva clave de administrador se ingresa y se salvan en el EPP.

Ingresar Clave de Transmisión	Habilitar el ingreso de una serie de dígitos que conformen la clave de Transmisión del primer y segundo administrador, controlado por un tiempo de expiración. El EPP informa la finalización de la operación a la aplicación Cliente. La nueva clave de transmisión se ingresa y se salvan en el EPP.
Descarga Clave Maestra	Descargar clave maestra <M_Key> número <M_Key_No> indicada por la aplicación cliente.
Descargar Clave de Autenticación	Descargar clave de autenticación <A_Key> número <A_Key_No> indicada por la aplicación cliente.
Descargar Clave de Trabajo	Descargar clave de trabajo <S_Key> número <S_Key_No> indicada por la aplicación cliente.

Tabla 4.3.1. – Funciones específicas del EPP

Función	Descripción
Status	Permite consultar la disponibilidad de ejecución de las funcionalidades del dispositivo EPP.
Capability	Permite conocer la capacidad de ejecución de las funcionalidades del dispositivo EPP.
Versión	Permite consultar la información general sobre el driver y el dispositivo EPP.

Tabla 4.3.2. – Funciones generales del EPP

4.4 Análisis del modelo de amenazas

El primer paso es determinar mediante un modelo de amenazas, los objetivos de seguridad que debe satisfacer la aplicación a construir. Una posibilidad, no necesariamente la única, es utilizar STRIDE/DREAD.

STRIDE/DREAD (Ver 2.4.4.1) permite al usuario visualizar escenarios potenciales de amenazas desde la perspectiva de un atacante, agrupándolos en categorías: "***S**poofing*" robo de identidad; "***T**ampering*" cambio no autorizado de datos; "***R**epudiation*" repudio de servicios o acciones; "***I**nformation disclosure*" divulgación de información; "***D**enial of service*" denegación de servicios y "***E**levation of privilege*" elevación de privilegio. Luego se aplica una metodología de valoración de riesgos, conocida como DREAD, cuyo nombre encapsula las respuestas a potenciales preguntas sobre los riesgos:

- Daño potencial "***D**amage potential*" : Qué tan grande es el daño si la vulnerabilidad es explotada..?
- Reproducibilidad "***R**eproducibility*" : Qué tan fácil es reproducir el ataque..?
- Aprovechable "***E**xploitability*" : Qué tan fácil es lanzar el ataque..?
- Usuarios afectados "***A**ffected users*" : En porcentaje, cuántos usuarios afectaría..?
- Descubrible "***D**iscoverability*" : Qué tan fácil es encontrar la vulnerabilidad..?

DREAD ayuda a valorar amenazas y priorizar la importancia de las contramedidas y acciones para mitigarlas. Una vez clasificados los atributos de las amenazas, se toma una media de los cinco atributos, lo que resulta en una valoración del riesgo percibido asociado a la amenaza. Este proceso es repetido para todas las amenazas identificadas y luego permite priorizarlas en orden descendente.

La evaluación se realiza sólo para el Driver, debido a que es el único producto que será pasado a producción en el terminal "*kiosk*". En la Tabla 4.4.1 se muestra el Nivel de gravedad utilizado para clasificar a las amenazas. Y luego se muestran las tablas de STRIDE para el Driver, evaluando Amenaza de Suplantación de Identidad (Tabla 4.4.2); Amenaza de Manipulación (Tabla 4.4.3); Amenaza de Repudio (Tabla 4.4.4); Amenaza de Revelación de información (Tabla 4.4.5); Amenaza de Denegación de servicio (Tabla 4.4.6) y Amenaza de Elevación de privilegios (Tabla 4.4.7).

Puntuación	Definición
Alta	Amenaza ocurre de manera inminente sobre un recurso crítico del sistema.
Moderada	Amenaza podría reducirse en gran medida mediante factores como una configuración predeterminada, auditoría o dificultad de abuso.
Baja	Amenaza muy difícil de aprovechar o cuyo impacto es mínimo.

Tabla 4.4.1. – Nivel de amenazas

Amenaza	*Suplantación de Identidad (Spoofing)*
Objetivo de amenaza	**Driver**
Descripción	Atacante intenta establecer enlace de comunicación con Driver.
Nivel de amenaza	**Baja.** Debido a que si el atacante estableciera comunicación con el Driver, necesita tener acceso a las claves del EPP para ejecutar cualquier comando.
Probabilidad de ocurrencia	**Baja.** Existen dos posibilidades. 1) Demanda que el atacante tenga acceso a la red de la entidad financiera. La infraestructura de red está protegida de acuerdo a los estándares del BCRA y a las políticas de seguridad de la propia entidad financiera. 2) Que el atacante tenga acceso físico al terminal kiosk y establezca una comunicación desde esa posición. Esto demanda una dificultad muy grande y una muy baja probabilidad de ocurrencia.
Mitigación	La comunicación del Driver (sólo tiene posibilidades de pasar información) es limitada y establecida sólo a la dirección ip:puerto definida como habilitada.

Tabla 4.4.2. – Amenaza de Suplantación

Amenaza	*Manipulación (Tampering)*
Objetivo de amenaza	**Driver**
Descripción	Atacante intenta modificar los datos de transferencia entre Driver y la entidad ACG y/o dispositivo EPP.
Nivel de amenaza	**Baja.** Debido a que para lograrlo, el atacante necesita tener acceso a las claves de cada funcionalidad del EPP para autorizar la ejecución de cada comando.
Probabilidad de ocurrencia	**Baja.** Es muy baja la probabilidad de que el atacante tenga acceso a las claves de transmisión de datos.
Mitigación	Todos los datos que traspasa el Driver se encuentran encriptados con 3DES y en ninguna circunstancia éste los desencripta.

Tabla 4.4.3. – Amenaza de Manipulación

Amenaza	*Repudio (Repudiation)*
Objetivo de amenaza	**Driver**
Descripción	Atacante niega la ejecución de acciones realizadas.
Nivel de amenaza	**Baja.** Cada persona es responsable de una sola clave de autorización de una única funcionalidad. Se necesita que el atacante tenga acceso a las claves de cada funcionalidad del EPP para ejecutar cada comando.
Probabilidad de ocurrencia	**Baja.** Se necesita que el atacante sea personal de la entidad financiera con acceso a todas las claves de cada funcionalidad del EPP.
Mitigación	El Driver funciona como un puente/pasamanos para la comunicación entre ACG/EPP. El diseño del dispositivo EPP cubre la identificación de acciones de cada responsable. La autoría de las acciones quedan individualizadas no solo con la dirección ip:puerto del terminal cliente habilitado sino que con las claves utilizadas para autorizar esas acciones se determina el administrador responsable.

Tabla 4.4.4. – Amenaza de Repudio

Amenaza	*Revelación de información (Information Disclosure)*
Objetivo de amenaza	**Driver**
Descripción	Atacante intenta revelar los datos de transferencia entre la entidad ACG y/o el dispositivo EPP.
Nivel de amenaza	**Baja.** El atacante podría revelar la información de transferencia pero necesita tener acceso a las claves de Administrador del EPP para ejecutar comandos.
Probabilidad de ocurrencia	**Baja.** Por las mismas razones consideradas para *Suplantación de Identidad*.
Mitigación	Todos los datos que traspasa el Driver se encuentran encriptados con 3DES y en ninguna circunstancia éste los desencripta.

Tabla 4.4.5. – Amenaza de Revelación de información

Amenaza	*Denegación de servicio (Denial of Service)*
Objetivo de amenaza	**Driver**
Descripción	Atacante intenta dejar fuera de servicio o bloquear acceso a Driver.
Nivel de amenaza	**Media.** El atacante podría dejar temporalmente sin servicios de Driver al terminal kiosk hasta que el Sistema Operativo detecte esta detención e inicie nuevamente el servicio.
Probabilidad de ocurrencia	**Baja.** Por las mismas razones consideradas para *Suplantación de Identidad*.
Mitigación	La red y terminal kiosk donde el Driver se despliega en producción se encuentran protegidas bajo las políticas de seguridad de la entidad financiera. El Driver es un Servicio de Sistema Operativo instalado para su ejecución automática cuando este se detiene.

Tabla 4.4.6. – Amenaza de Denegación de servicio

Amenaza	*Elevación de privilegios (Elevation of Privilege)*
Objetivo de amenaza	**Driver**
Descripción	Atacante podría utilizar Driver para obtener permisos de Administrador y ejecutar código.
Nivel de amenaza	**Baja.** Debido a que el atacante se encuentra limitado al proceso asociado al Driver (aislación de procesos).
Probabilidad de ocurrencia	**Baja.** Por las mismas razones consideradas para *Suplantación de Identidad*.
Mitigación	Las políticas de seguridad de la entidad financiera utilizan mecanismos de aislación de procesos de Windows y se considera suficiente.

Tabla 4.4.7. – Amenaza de Elevación de Privilegios

De la evaluación de amenazas, se concluye que si un atacante plantea como objetivo el Driver, debe hacer un ataque en profundidad, superando la protección brindada por la infraestructura de comunicaciones. La infraestructura de comunicaciones está protegida de acuerdo a los estándares del Banco Central de la República Argentina (BCRA) y a las Políticas de Seguridad de la propia entidad financiera que no son motivos de discusión en este trabajo. Suponiendo que el atacante pueda superar esa barrera, se encontrará con una aplicación que lo único que hace es pasar información hacia y desde el EPP. Esa información se encuentra protegida mediante un servicio de confidencialidad utilizando el algoritmo triple DES (3DES), que para el contexto se considera seguro. De poder superar la protección del proceso de encriptación con 3DES, deberá conocer las claves de acceso a cualquiera de las funcionalidades del EPP que están distribuidas entre varios usuarios.

Conclusión: se puede confiar en las mitigaciones propuestas para minimizar la vulnerabilidad del Driver.

Ahora bien, para garantizar la seguridad en la construcción del Driver, se procede a elaborar un modelo de requerimientos, que tenga en cuenta las mejores prácticas que permitan minimizar los riesgos desde las etapas tempranas de elicitación y de análisis de requerimientos. Para ello se propone modelar los requerimientos de la aplicación, incorporando patrones de seguridad, lo cual se materializará en la Sección 4.5.

4.5 Modelo de requerimientos

4.5.1 Construcción del LEL de la Aplicación

Se extraen nueve signos del dominio de la aplicación. Las entradas del LEL son "Parámetros de Ejecución" (Tabla 4.5.1); "Aplicación Cliente Genérica (ACG)" (Tabla 4.5.2); "EPP" (Tabla 4.5.3); "Comando/Comandos" (Tabla 4.5.4); "Tabla de Procesos del SO" (Tabla 4.5.5); "Tabla de Servicios del SO" (Tabla 4.5.6); "Registro del SO" (Tabla 4.5.7); "Driver / Controlador de Dispositivo" (Tabla 4.5.8) y "Encripta / desencripta" (Tabla 4.5.9).

Sinónimos	Parámetro de Ejecución
Noción	Entidad utilizada para seleccionar la tarea a ejecutar por el Driver
Impacto	Permite identificar los estados del Driver

Tabla 4.5.1 – Entrada de LEL "Parámetro de Ejecución"

Sinónimos	Aplicación Cliente Genérica (ACG)
Noción	Aplicación que solicita servicios al Driver
Impacto	Envía Comando al Driver para solicitar servicios del EPP. Encripta / desencripta información. Genera nonce. Conoce Claves de Transmisión. Descarga Clave Maestra, de Autenticación y de Trabajo en el EPP.

Tabla 4.5.2 – Entrada de LEL "Aplicación Cliente Genérica (ACG)"

Sinónimos	EPP
Noción	Dispositivo que solicita servicios del Driver para que ACG pueda Validar Clave de Administrador, Ingresar Clave de Transmisión. Dispositivo utilizado para Ingresar PIN e Ingresar Dígitos.
Impacto	Ejecuta Comando enviado por ACG utilizando Driver. Envía Comando al Driver para responder peticiones de servicios del ACG. Encripta /desencripta información y genera nonce. Conoce Claves de Administrador y Claves de Transmisión.

Tabla 4.5.3 – Entrada de LEL "EPP"

Sinónimos	Comando / Comandos
Noción	Entidad utilizada por ACG para solicitar servicios del Driver Entidad utilizada por EPP para responder peticiones de servicios del ACG
Impacto	Es enviado por ACG a Driver para utilizar servicios del EPP. Es enviado por EPP a Driver para comunicarse con ACG.

Tabla 4.5.4 – Entrada de LEL "Comando / Comandos"

Sinónimos	Tabla de Procesos de SO
Noción	Entidad del SO que contiene información sobre los Procesos en ejecución.
Impacto	Agrega nueva entrada cuando el Driver se Inicia. Elimina entrada que corresponde al Driver, cuando el Driver se detiene.

Tabla 4.5.5 – Entrada de LEL "Tabla de Procesos de SO"

Sinónimos	Tabla de Servicios de SO
Noción	Entidad del SO que contiene información sobre los Servicios que brinda el SO.
Impacto	Agrega nueva entrada cuando el Driver se Instala. Elimina entrada que corresponde al Driver, cuando el Driver se Elimina.

Tabla 4.5.6 – Entrada de LEL "Tabla de Servicios de SO"

Sinónimos	Registro de SO
Noción	Base de datos con información de configuración del SO.
Impacto	Almacena Parámetros de Ejecución del Driver, cuando se Instala. Driver lee Parámetros de Ejecución del Registro de SO cuando se Inicia. Elimina Parámetros de Ejecución del Driver, cuando se Elimina.

Tabla 4.5.7 – Entrada de LEL "Registro de SO"

Sinónimos	Driver
Noción	Es el intermediario entre ACG y EPP.
Impacto	Recibe Comando enviado por ACG y se lo entrega al EPP. Recibe Comando enviado por EPP y lo entrega a ACG.

Tabla 4.5.8 – Entrada de LEL "Driver/Controlador de Dispositivo"

Sinónimos	Encripta / Desencripta
Noción	Acción por la cual ACG/EPP oculta al Driver la información enviada al EPP/ACG, utilizando algoritmo 3DES y una Clave
Impacto	A la comunicación entre ACG y EPP se le aplica un servicio de confidencialidad.

Tabla 4.5.9 –Entrada de LEL "Encripta/Desencripta"

4.5.2 Modelo de casos de uso de Driver y ACG

Para la construcción de los escenarios, se recurre a la construcción de los casos de uso para el Driver y para ACG. Los escenarios son una representación textual directa de cada caso de uso. En la Figura 4.5.2.1 se muestra el diagrama de casos de uso para el Driver.

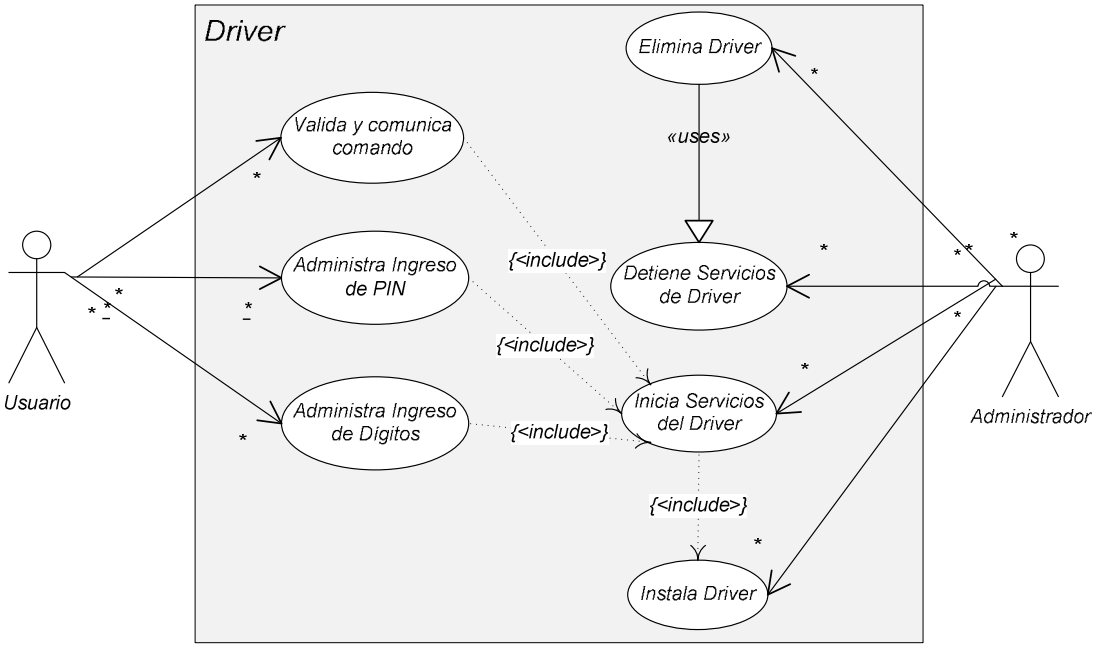

Figura 4.5.2.1 – Diagrama de casos de uso para Driver

En la Figura 4.5.2.2 se muestra el diagrama de casos de uso para la Aplicación Cliente Genérica (ACG).

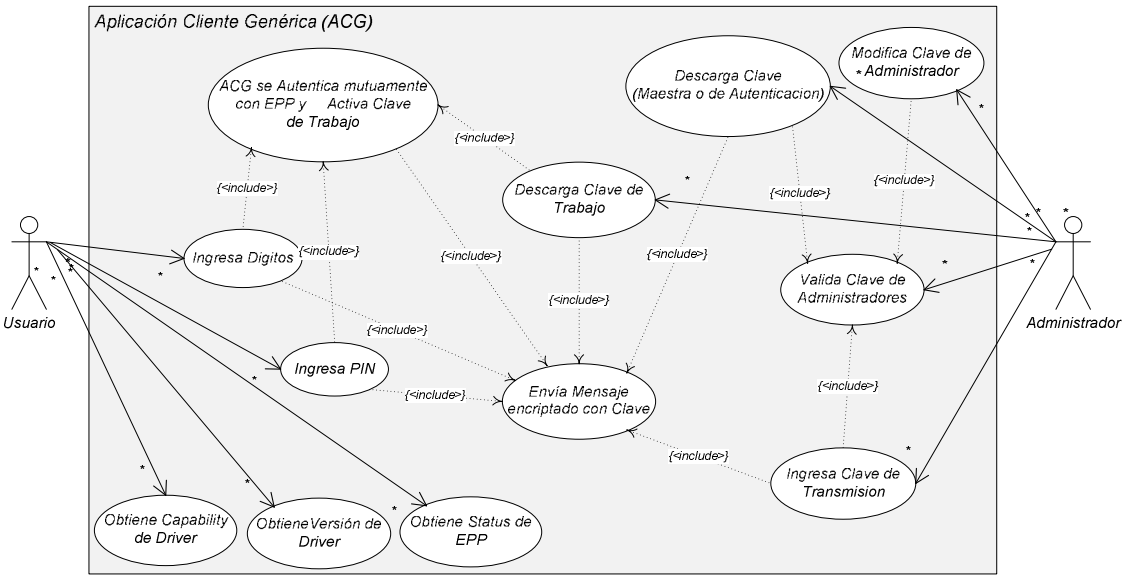

Figura 4.5.2.2 – Diagrama de casos de uso para ACG

4.5.3 Construcción de Escenarios a partir de los casos de uso

A continuación se describe cada caso de uso del Driver, como un escenario. Se muestra "Instala Driver" (Tabla 4.5.3.1); "Inicia Servicios del Driver" (Tabla 4.5.3.2); "Detiene Servicios del Driver" (Tabla 4.5.3.3); "Elimina Driver" (Tabla 4.5.3.4); "Valida y comunica comando" (Tabla

4.5.3.5); “Administra Ingreso de PIN” (Tabla 4.5.3.6); “Administra Ingreso de Dígitos” (Tabla 4.5.3.7).

Título	Instala Driver
Objetivo	Agregar Driver a la Tabla de Servicios del SO
Contexto	PC encendida
Recursos	Driver, Parámetros de Ejecución
Actores	Administrador
Episodios	1. Administrador ejecuta Driver con Parámetros de Ejecución (Instalación). 2. Tabla de Servicios de SO agrega una nueva entrada a su estructura. 3. Registro de SO almacena Parámetros de Ejecución del Driver.

Tabla 4.5.3.1 – Escenario “Instala Driver”

Título	Inicia Servicios del Driver
Objetivo	Crear un nuevo proceso que brinde los Servicios del Driver
Contexto	Driver instalado
Recursos	Driver, Parámetro de Ejecución
Actores	Administrador
Episodios	1. Administrador ejecuta Driver con Parámetro de Ejecución (Inicia). 2. Driver obtiene Parámetro de Ejecución (Instalación) del Registro del SO 3. Tabla de Proceso de SO agrega una nueva entrada a su estructura.

Tabla 4.5.3.2 – Escenario “Inicia Servicios del Driver”

Título	Detiene Servicios de Driver
Objetivo	Eliminar entrada del Driver de Tabla de Procesos del SO
Contexto	Driver iniciado
Recursos	Driver, Parámetros de Ejecución
Actores	Administrador
Episodios	1. Administrador ejecuta Driver con Parámetros de Ejecución (detención). 2. Tabla de Proceso de SO elimina de su estructura la entrada que corresponde al Driver.

Tabla 4.5.3.3. – Escenario “Detiene Servicios del Driver”

Título	Elimina Driver
Objetivo	Eliminar entrada correspondiente al Driver de la Tabla de Servicios de SO
Contexto	Servicios del Driver detenidos
Recursos	Driver, Parámetro de Ejecución
Actores	Administrador
Episodios	1. Administrador ejecuta Driver con Parámetros de Ejecución (elimina). 2. Registro de SO elimina Parámetros de Ejecución del Driver. 3. Tabla de Servicios de SO elimina de su estructura, la entrada que corresponde al Driver.

Tabla 4.5.3.4. – Escenario “Elimina Driver”

Título	Valida Traduce y Comunica comando
Objetivo	Ejecutar un comando enviado por ACG al EPP
Contexto	Comunicación establecida entre ACG y EPP, Driver iniciado
Recursos	Comando
Actores	ACG, EPP, Driver
Episodios	1. ACG envía al Driver Comando para ejecutar en el EPP. 2. Driver valida Comando. 3. Si Comando es NO VALIDO, Driver envía Comando de finalización de proceso a ACG. 4. Si Comando es VALIDO, Driver traduce Comando y lo envía a EPP. 5. EPP ejecuta Comando y envía resultado de ejecución a Driver. 6. Driver informa resultado a ACG.

Tabla 4.5.3.5. – Escenario "Valida y comunica comando"

Título	Administra Ingreso de PIN
Objetivo	Controlar el proceso de ingreso de PIN
Contexto	Comunicación establecida entre ACG y EPP, Driver iniciado
Recursos	Comando para Ingreso de PIN
Actores	ACG, EPP, Driver
Episodios	1. ACG envía comando para Ingreso de PIN a Driver. 2. Driver valida traduce y comunica Comando a EPP. 3. Si Driver recibe del EPP un dígito oculto (0x2A) incrementa contador en uno y pasa dígito oculto al ACG. 4. Si Driver recibe del EPP un CORREGIR reinicia contador. 5. Si Driver recibe del EPP un PIN BLOCK envía PIN BLOCK y comando de finalización exitosa de operación a ACG. 6. Si Driver recibe del EPP un CANCEL o un TIMEOUT, envía comando de finalización de operación a ACG.

Tabla 4.5.3.6 – Escenario "Administra Ingreso de PIN"

Título	Administra Ingreso de Dígitos
Objetivo	Controlar el proceso de ingreso de dígitos individuales
Contexto	Comunicación establecida entre ACG y EPP, Driver iniciado
Recursos	Comando para Ingreso de Dígitos
Actores	ACG, EPP, Driver
Episodios	1. ACG envía comando para ejecutar Ingreso de Dígitos a Driver. 2. Driver valida traduce y comunica Comando a EPP 3. Si Driver recibe de EPP un dígito, se lo pasa a ACG. 4. Si Driver recibe de EPP un CANCEL, envía comando de finalización de operación a ACG. 5. Si Driver recibe de EPP un ENTER, envía comando de finalización exitosa de operación a ACG.

Tabla 4.5.3.7. – Escenario "Administra Ingreso de Dígitos"

A continuación se describe cada caso de uso del ACG, como un escenario. Se muestra "Valida Clave de Administradores" (Tabla 4.5.3.8); "Modificar Clave de Administrador número Adm_Key_No" (Tabla 4.5.3.9); "Ingresar Clave de Transmisión" (Tabla 4.5.3.10); "Descarga Clave (Maestra o de Autenticación)" (Tabla 4.5.3.11); "ACG autentica mutuamente EPP y activa Clave de Trabajo núm. S_Key_Nro" (Tabla 4.5.3.12); "Descarga Clave de Trabajo número S_Key_Nro" (Tabla 4.5.3.13); "Ingresa PIN" (Tabla 4.5.3.14); "Ingresa Dígitos" (Tabla 4.5.3.15); "Obtiene Status del EPP" (Tabla 4.5.3.16); "Obtiene versión del Driver" (Tabla 4.5.3.17); "Obtiene Capability del Driver" (Tabla 4.5.3.18) y "Envía Mensaje encriptado con Clave" (Tabla 4.5.3.19).

Título	Valida Clave de Administradores
Objetivo	Aceptar o rechazar Clave de Administradores
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando para ingreso de Clave de Administradores
Actores	ACG, EPP, Driver, Administrador
Episodios	1. ACG envía a EPP comando para ingreso de Clave de Administradores. 2. EPP emite un beep largo y espera ingreso de primer Clave de Administrador. 3. Administrador ingresa primer Clave de Administrador. 4. Si primer Clave de Administrador es incorrecta, EPP emite dos beep cortos y envía comando de finalización de proceso a ACG. 5. Si primer Clave de Administrador es correcta, EPP emite un beep largo y espera ingreso de segunda Clave de Administrador. 6. Administrador ingresa segunda Clave de Administrador. 7. Si segunda Clave de Administrador es incorrecta, EPP emite dos beep cortos y envía comando de finalización de proceso a ACG. 8. Si segunda Clave de Administrador es correcta, EPP emite un beep largo y envía comando de aceptación de Claves de Administrador a ACG.

Tabla 4.5.3.8. – Escenario "Valida Clave de Administradores"

Título	Modificar Clave de Administrador número Adm_Key_No
Objetivo	Salvar una nueva Clave de Administrador
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando para modificación de Clave de Administrador número Adm_Key_No
Actores	ACG, EPP, Driver, Administrador
Episodios	1. ACG envía a EPP comando para modificación de Clave de Administrador número Adm_Key_No. 2. EPP emite un beep largo y espera ingreso de Clave de Administrador número Adm_Key_No. 3. Administrador ingresa Clave de Administrador número Adm_Key_No. 4. Si Clave de Administrador número Adm_Key_No es incorrecta, EPP emite dos beep cortos y envía comando de finalización de proceso a ACG. 5. Si Clave de Administrador número Adm_Key_No es correcta, EPP emite un beep largo y espera ingreso de nueva Clave de Administrador número Adm_Key_No. 6. Administrador ingresa por primera vez nueva Clave de Administrador número Adm_Key_No.

	7. Administrador ingresa por segunda vez nueva Clave de Administrador número Adm_Key_No. 8. Si nueva Clave de Administrador se ingresó incorrectamente, EPP emite dos beep cortos y finaliza comando de modificación de Clave de Administrador número Adm_Key_No. 9. Si nueva Clave de Administrador se ingresó correctamente, EPP salva nueva Clave de Administrador número Adm_Key_No, emite un beep largo y envía comando de finalización exitosa de proceso a ACG.

Tabla 4.5.3.9. – Escenario "Modificar Clave de Administrador número Adm_Key_No"

Título	Ingresa Clave de Transmisión
Objetivo	Aceptar o rechazar Claves de Transmisión
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando para ingreso de Claves de Transmisión
Actores	ACG, EPP, Driver
Episodios	1. ACG Valida Clave de Administradores exitosamente. 2. ACG envía a EPP comando para Ingreso de Claves de Transmisión. 3. EPP emite un beep largo y espera ingreso de primer Clave de Transmisión. 4. Administrador ingresa primer Clave de Transmisión. 5. Si primer Clave de Transmisión es incorrecta, EPP emite dos beep cortos y envía comando de finalización de proceso a ACG. 6. Si primer Clave de Transmisión es correcta, EPP emite un beep largo, envía a ACG comando de finalización de primer Clave de Transmisión y espera ingreso de segunda Clave de Transmisión. 7. Administrador ingresa segunda Clave de Transmisión. 8. Si segunda Clave de Transmisión es incorrecta, EPP emite dos beep cortos y envía comando de finalización de proceso a ACG. 9. Si segunda Clave de Transmisión es correcta, EPP emite un beep largo y envía comando de aceptación de Claves de Transmisión a ACG. 10. ACG recibe comando e informa que proceso ha finalizado exitosamente.

Tabla 4.5.3.10. – Escenario "Ingresar Clave de Transmisión"

Título	Descarga Clave (Maestra o de Autenticación)
Objetivo	Salvar Clave en el EPP
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando para ingreso de Clave (Maestra o de Autenticación), Claves de Transmisión
Actores	ACG, EPP, Driver
Episodios	1. ACG Valida Clave de Administradores exitosamente. 2. ACG envía Mensaje (Clave Maestra o de Autenticación) encriptado con Clave (Clave de Transmisión) a EPP 3. EPP salva Clave. 4. EPP informa a ACG que la operación ha finalizado exitosamente.

Tabla 4.5.3.11. – Escenario "Descarga Clave (Maestra o de Autenticación)"

Título	ACG se autentica mutuamente con EPP y activa Clave de Trabajo
Objetivo	Verificar mutuamente identidad entre ACG y EPP y Activar Clave de Trabajo
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando para inicio de Autenticación mutua entre ACG y EPP, Clave de Autenticación número A_Key_Nro
Actores	ACG, EPP, Driver
Episodios	1. ACG genera nonce N_H. 2. ACG envía a EPP comando de Petición de Inicio de Autenticación junto a A_Key_Nro, S_Key_Nro y N_H. 3. EPP envía Mensaje (N_H) encriptado con Clave (Clave de Autenticación número A_Key_Nro) a ACG. 4. ACG compara N'_H con N_H y si son iguales envía comando de finalización exitosa de autenticación a EPP. 5. EPP genera nonce N_K. 6. EPP envía a ACG comando de Petición de Inicio de Autenticación junto a A_Key_Nro, S_Key_Nro y N_K. 7. ACG envía Mensaje (N_K) encriptado con Clave (Clave de Autenticación número A_Key_Nro) a EPP. 8. EPP compara N'_K con N_K y si son iguales envía comando de finalización exitosa de autenticación a ACG.

Tabla 4.5.3.12. – Escenario "ACG autentica mutuamente EPP y activa Clave de Trabajo núm. S_Key_Nro"

Título	Descarga Clave de Trabajo número S_Key_Nro
Objetivo	Salvar Clave de Trabajo número S_Key_Nro en el EPP
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando para ingreso de Clave de Trabajo número S_Key_Nro, Claves Maestra número M_Key_Nro
Actores	ACG, EPP, Driver
Episodios	1. ACG autentica mutuamente con EPP y activa Clave de Trabajo número S_Key_Nro. 2. ACG envía Mensaje (Clave de Trabajo número S_Key_Nro) encriptado con Clave (Clave Maestra M_Key_Nro) a EPP 3. EPP guarda Clave de Trabajo número S_Key_Nro. 4. EPP informa a ACG que la operación ha finalizado exitosamente.

Tabla 4.5.3.13. – Escenario "Descarga Clave de Trabajo número S_Key_Nro"

Título	Ingresa PIN
Objetivo	Ingresar un PIN utilizando el EPP
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando para Ingreso de PIN, Clave Trabajo número S_Key_Nro, Algoritmo 3DES
Actores	Usuario, ACG, EPP y Driver
Episodios	1. ACG autentica mutuamente con EPP y activa Clave de Trabajo número S_Key_Nro. 2. ACG envía a EPP comando para Ingreso de PIN. Contiene la longitud de PIN válido y tiempo de espera máximo de ingreso de un dígito (TIMEOUT). 3. EPP espera ingreso de primer digito. 4. Usuario ingresa digito en el EPP. 5. EPP envía digito oculto (0x2A) a ACG. 6. Si Usuario presionó la tecla CANCEL o se cumplió TIMEOUT, EPP envía código de finalización de proceso de Ingreso de PIN a ACG. 7. Si Usuario ingresa PIN completo, EPP envía Mensaje (PinBlock) encriptado con Clave (Clave de Trabajo número S_Key_Nro) a la ACG. 8. EPP envía comando de finalización exitosa de operación a ACG.

Tabla 4.5.3.14. – Escenario "Ingresa PIN"

Título	Ingresa Dígitos
Objetivo	Ingresar dígitos individuales utilizando el EPP
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando para Ingreso de Dígitos, Clave Trabajo número S_Key_Nro, Algoritmo 3DES
Actores	Usuario, ACG, EPP, Driver
Episodios	1. ACG autentica mutuamente con EPP y activa Clave de Trabajo número S_Key_Nro. 2. ACG envía a EPP comando para Ingreso de Dígitos. 3. EPP espera ingreso de uno o más dígitos. 4. Usuario ingresa un dígito utilizando el EPP. 5. EPP envía Mensaje (dígito) encriptado con Clave (Clave de Trabajo número S_Key_Nro) a la ACG. 6. Si Usuario no presiona CANCEL ni ENTER, EPP comunica a ACG que continúa en espera de ingreso de dígito. 7. Si Usuario presiona CANCEL, EPP envía comando de finalización NO exitosa de operación a ACG. 8. Si Usuario presiona ENTER, EPP envía comando de finalización exitosa de operación a ACG.

Tabla 4.5.3.15. – Escenario "Ingresa Dígitos"

Título	Obtiene Status del EPP
Objetivo	Conocer el estado actual del EPP
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando de Status
Actores	ACG, EPP y Driver
Episodios	1. ACG envía comando de solicitud de Status del EPP a Driver. 2. Driver recibe comando y comprueba estado de disponibilidad del EPP 3. Driver comunica disponibilidad/indisponibilidad de EPP a ACG.

Tabla 4.5.3.16. – Escenario "Obtiene Status del EPP"

Título	Obtiene versión del Driver
Objetivo	Conocer versión del Driver en ejecución
Contexto	Driver ejecutado y comunicación establecida entre ACG y Driver
Recursos	Comando de Versión
Actores	ACG, Driver
Episodios	1. ACG envía comando de Versión a Driver. 2. Driver recibe comando de Versión e informa versión de Driver en ejecución a ACG.

Tabla 4.5.3.17. – Escenario "Obtiene versión del Driver"

Título	Obtiene Capability del Driver
Objetivo	Conocer Capability del Driver en ejecución
Contexto	Driver ejecutado y comunicación establecida entre ACG y Driver
Recursos	Comando de Capability
Actores	ACG, Driver
Episodios	1. ACG envía comando de Capability a Driver. 2. Driver recibe comando de Capability e informa capacidad de ejecución de Driver en ejecución a ACG.

Tabla 4.5.3.18. – Escenario "Obtiene Capability del Driver"

Título	Envía Comando encriptado con Clave
Objetivo	Ocultar el contenido del Comando
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando, Clave
Actores	ACG, EPP
Episodios	1. ACG encripta Comando con Clave y se lo envía a Driver. 2. Driver envía Comando Encriptado a EPP. 3. EPP desencripta Comando Encriptado con Clave. 4. EPP ejecuta Comando.

Tabla 4.5.3.19. – Escenario "Envía Comando encriptado con Clave"

4.5.4 Obtención de tarjetas CRC

Para deducir las CRC, a partir de entradas del LEL y Escenarios se utilizó la herramienta automatizada BMW. En un principio de utilizó el BMW v1.0, construido por Antonelli, luego se implementó una nueva versión que mantiene idéntica funcionalidad y permite aislar el modelo de Patrones de Seguridad. El BMW v2.0, el cual fue desarrollado exclusivamente a los propósitos de este trabajo de Maestría.

Se obtuvieron las CRC "*ACG*" (Tabla 4.5.4.1); "*Comando*" (Tabla 4.5.4.2) "*Driver*" (Tabla 4.5.4.3) y "*EPP*" (Tabla 4.5.4.4).

Primary CRC Card	ACG
Responsabilities	Envía Comando al Driver para solicitar servicios del EPP. Encripta/Desencripta información. Genera nonce. Conoce Claves de Transmisión. Descarga Clave Maestra, de Autenticación y de Trabajo en el EPP.
Collaborations	COMANDO, DRIVER, EPP, ADMINISTRADOR

Tabla 4.5.4.1 – CRC Primaria "ACG"

Secondary CRC Card	Comando
Responsabilities	Es enviado por ACG a Driver para utilizar servicios del EPP. Es enviado por EPP a Driver para comunicarse con ACG.
Collaborations	ACG, EPP, DRIVER, ADMINISTRADOR

Tabla 4.5.4.2 – CRC Secundaria "Comando"

Primary CRC Card	Driver
Responsabilities	Recibe Comando enviado por ACG y se lo entrega al EPP. Recibe Comando enviado por EPP y lo entrega a ACG.
Collaborations	ACG, ADMINISTRADOR, COMANDO, EPP

Tabla 4.5.4.3. – CRC Primaria "Driver"

Primary CRC Card	EPP
Responsabilities	Ejecuta Comando enviado por ACG utilizando el Driver. Envía Comando al Driver para responder peticiones de servicios del ACG. Encripta /desencripta información y genera nonce. Conoce Claves de Administrador y Claves de Transmisión.
Collaborations	ACG, ADMINISTRADOR, COMANDO, DRIVER

Tabla 4.5.4.5. – CRC Primaria "EPP"

4.5.5 Primer modelo estático de objetos de la realidad

A partir de las CRC deducidas y sus relaciones, se construye el diagrama de objetos de la realidad que se muestra en la Figura 4.5.5.1, que modela la aplicación que se pretende construir. El diagrama propone el uso de los objetos "ACG", "Driver", "EPP" y "Comando".

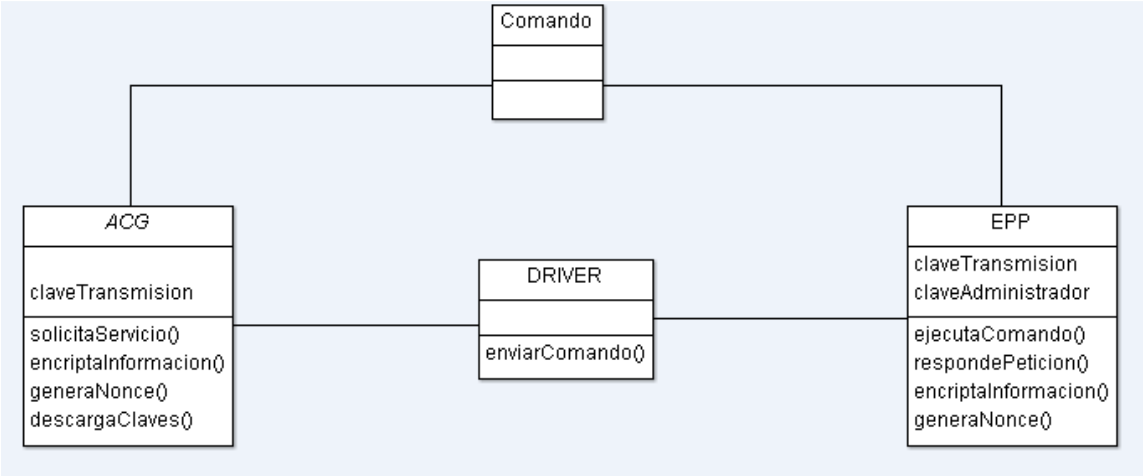

Figura 4.5.5.1. – Diagrama de objetos de la realidad

Se puede observar que entre las responsabilidades de ACG y EPP, está llevar adelante un proceso de encriptación de información, lo que define claramente un objetivo de seguridad para brindar servicios de confidencialidad.

4.5.6 Aproximación a la identificación del Patrón de Seguridad

Cómo proceder a utilizar un Patrón de Seguridad y no otro..? Cómo mapear los requerimientos de seguridad de una aplicación sobre un Patrón de Seguridad..? En esta sección se describe el proceso de identificación del Patrón de Seguridad entre los requerimientos de la aplicación que se desea construir.

Si tenemos en cuenta que en el Capítulo 3, se modeló el Patrón de Seguridad, GOOCA utilizando LEL y Escenarios. Y que en la descripción del patrón (ver 3.5.5.3) se menciona que el mismo es una buena solución para el siguiente problema: "*Cómo diseñar una micro-arquitectura orientada a objetos para un diseño criptográfico y facilitar la reutilización de estos componente.*". Y que la misma documentación del patrón sugiere en **Aplicabilidad** lo siguiente:

- *Cuando se deben cumplir aspectos que se presentan muy separados entre requerimientos criptográficos no funcionales y requerimientos funcionales de una aplicación específica.*
- *Cuando se debe lograr una transformación criptográfica exitosa y la aplicación a desarrollar debe ser independiente de aquella transformación.*
- *Cuando es necesaria una interface genérica para varios tipos de servicios criptográficos.*

Se concluye que una muy buena propuesta para cumplir con el objetivo de seguridad "Confidencialidad", será utilizar este patrón de seguridad.

Una vez identificado el Patrón de Seguridad, se presenta el problema de tomar las mejores decisiones en la etapa de elicitación y análisis de requerimientos para integrar el patrón en un modelo que permita estudiar y elaborar propuestas para las etapas de diseño. Como estamos utilizando un

modelo contextual construido con LEL y Escenarios, la mejor decisión a fin de mantener la consistencia sería incluir al Patrón de Seguridad en dicho modelo. Para ello necesitamos que dicho patrón tenga una representación en términos de LEL y Escenarios. Este patrón la tiene.

Cuando se modeló el Patrón de Seguridad en el capítulo anterior, se obtuvieron los términos de LEL y Escenario que modela dicho patrón (ver 3.6 y 3.7). A partir de ellos, se pudo obtener un conjunto de CRC que modelan las clases de diseño del patrón. En este caso ya no son objetos de la realidad, sino que representan un modelo abstracto estático UML con propuestas que deben implementarse mediante clases, si se quiere utilizar el patrón de seguridad en la implementación de la aplicación. En este contexto y con estos elementos, la incorporación del Patrón de Seguridad al modelo, condiciona fuertemente las decisiones que de aquí en adelante se puedan tomar, teniendo como horizonte la etapa de diseño e implementación de la aplicación. Por ello, el modelo completo de la aplicación estará fuertemente impulsado, traccionado, hacia dichas etapas. En esta medida el Patrón de Seguridad "conduce" al diseño de ahora en más.

Para mostrar esto, se presenta nuevamente a continuación, el conjunto de CRC obtenidas con la herramienta BMW y que representan al Patrón de Seguridad GOOCA. CRC primarias: "Alice" (Tabla 4.5.6.1), "Bob" (Tabla 4.5.6.2) y secundarias: "Codificador" (Tabla 4.5.6.3), y "Decodificador" (Tabla 4.5.6.4).

Primary CRC Card	*Alice*
Responsabilities	Solicita al Codificador encriptar el mensaje con la Clave. Envía Mensaje encriptado a Bob. Conoce Clave.
Collaborations	*BOB, CODIFICADOR, DECODIFICADOR*

Tabla 4.5.6.1. – CRC primaria "Alice"

Primary CRC Card	*Bob*
Responsabilities	Solicita al Decodificador desencriptar el mensaje encriptado con la Clave. Conoce Clave.
Collaborations	*CODIFICADOR, DECODIFICADOR, ALICE*

Tabla 4.5.6.2. – CRC primaria "Bob"

Secondary CRC Card	*Codificador*
Responsabilities	Conoce Algoritmo_de_Encriptación.
Collaborations	*ALICE, BOB, DECODIFICADOR*

Tabla 4.5.6.3. – CRC secundaria "Codificador"

Secondary CRC Card	*Decodificador*
Responsabilities	Conoce Algoritmo_de_Encriptación.
Collaborations	*BOB, ALICE, CODIFICADOR*

Tabla 4.5.6.4. – CRC secundaria "Decodificador"

En la Figura 4.5.6.1 se muestra el diagrama de clases que generaliza la transformación criptográfica en un interface abstracta y separa los roles de emisor/receptor (Alice) y receptor/emisor (Bob) de los roles de Codificador/Decodificador y Decodificador/Codificador. GOOCA es una abstracción de alto nivel para brindar más de un servicio de seguridad. GOOCA tiene dos clases, Alice y Bob, que representan el "todo" de una relación de agregación; y dos clases asociadas que representan la "parte". Ellas son "*Codifier*" y "*Decodifier*". La clase "*Codifier*" tiene un método asociado **f()**, el cual realizar la transformación criptográfica sobre **m**. La clase "*Decodifier*" tiene su método asociado **g()**, el cual realiza la transformación criptográfica sobre **x = f(m)**. La transformación y su inversa están basadas sobre el mismo algoritmo criptográfico.

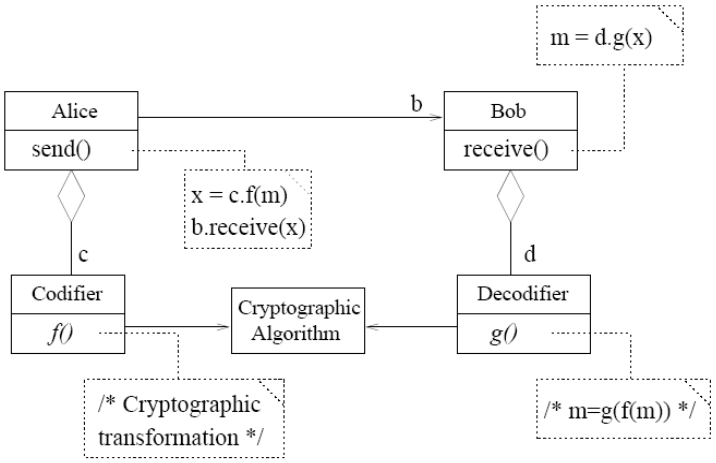

Figura 4.5.6.1. – Estructura estática de GOOCA

De ahora en adelante con estos dos elementos: el modelo contextual del Patrón de Seguridad por un lado y su modelo estático documentado por el otro, los que guíen las decisiones de análisis primero y de diseño después, de gran parte de la aplicación.

4.5.7 Integración de modelos estáticos

Es evidente la similitud entre el modelo estático de GOOCA mostrado en la Figura 4.5.6.1 y el diagrama de objetos de la realidad mostrado en la Figura 4.5.5.1. Esta última representa la aproximación al diseño del conjunto ACG + Driver + EPP. Ambas están relacionadas con un modelo contextual construido a partir de requerimientos utilizando LEL y Escenarios y deduciendo CRC con la herramienta BMW.

El objetivo en este punto es considerar las similitudes entre objetos de la realidad y clases de diseño propuestos por ambos modelos para redefinir o modificar aquellos términos de LEL y elementos de los Escenarios de manera que puedan integrarse los modelos del conjunto ACG + Driver + EPP y Patrón de Seguridad seleccionado (GOOCA). Al no abandonar los elementos que utilizamos en la construcción de ambos modelos, será posible continuar trabajando con la herramienta BMW v2.0, para deducir el conjunto de CRC que representarán a la aplicación, definitivamente con el Patrón de Seguridad embebido entre sus artefactos.

4.5.7.1 Análisis de CRC Primarias

Se examina en una primera instancia las CRC Primarias, recordando que sus responsabilidades son los impactos de los términos del LEL, identificados como actores en los Escenarios. Se toma

primero el término de LEL "*Alice*", descripto arriba. Se presenta nuevamente en la Tabla 4.5.7.1. A continuación se presenta el término de LEL ACG ya descripto, en la Tabla 4.5.7.2.

Sinónimo	Alice
Noción	Entidad que necesita mantener una comunicación confidencial con Bob
Impacto	Solicita al Codificador encriptar el mensaje con la Clave. Envía Mensaje encriptado a Bob. Conoce Clave.

Tabla 4.5.7.1. – Entrada de LEL "Alicia"

Sinónimos	Aplicación Cliente Genérica (ACG)
Noción	Aplicación que solicita servicios al Driver
Impacto	Envía Comando al Driver para solicitar servicios del EPP. Encripta / desencripta información. Genera nonce. Conoce Claves de Transmisión. Descarga Clave Maestra, de Autenticación y de Trabajo en el EPP.

Tabla 4.5.7.2. – Entrada de LEL "Aplicación Cliente Genérica (ACG)"

Analizando detenidamente los impactos de Alice, se ve que están incluidos entre los impactos de ACG. Los impactos de ACG, incluyen una tarea que Alice realiza en colaboración con Codificador y en conocimiento de Clave: Encripta/desencripta información. El impacto "enviar mensaje" ya está descripto en ACG. Por otro lado, Alice es una abstracción que representa uno de los extremos de una comunicación que cumple con un objetivo de seguridad: confidencialidad. En la aplicación que se desea construir ese extremo se encuentra representado por ACG. Por otro lado, la "Noción" de ACG, si bien no incluye una descripción que muestre la intencionalidad de mantener una comunicación con otro extremo, está implícita en el modelo. Esta intencionalidad está claramente expresada en la "Noción" de Alice. Hechas estas consideraciones se construye el nuevo término de LEL (ver Tabla 4.5.7.3) el cual mantiene el nombre de ACG e incluye la "Noción" y los "Impactos" de Alice en un nuevo contexto.

Sinónimos	Aplicación Cliente Genérica (ACG)
Noción	Aplicación que solicita servicios al Driver. Aplicación que necesita mantener una comunicación confidencial con EPP.
Impacto	Solicita al Codificador encriptar Comando con la Clave. Envía Comando encriptado al Driver. Genera nonce. Conoce Claves de Transmisión. Descarga Clave Maestra, de Autenticación y de Trabajo en el EPP.

Tabla 4.5.7.3. – Nueva entrada de LEL "Aplicación Cliente Genérica (ACG)"

Similares consideraciones se hacen sobre el par de términos de LEL, Bob, descripto en el Capítulo 3 y EPP, descripto en arriba en este Capítulo. Ambos se muestran en las Tablas 4.5.7.4 y

4.5.7.5 respectivamente. Luego se construye el nuevo término de LEL, EPP, descripto en la Tabla 4.5.7.6.

Sinónimo	Bob
Noción	Entidad que necesita mantener una comunicación confidencial con Alice.
Impacto	Solicita al Decodificador desencriptar el mensaje encriptado con la Clave. Conoce Clave.

Tabla 4.5.7.4. – Entrada de LEL "Bob"

Sinónimos	EPP
Noción	Dispositivo que solicita servicios del Driver para que ACG pueda Validar Clave de Administrador, Ingresar Clave de Transmisión. Dispositivo utilizado para Ingresar PIN e Ingresar Dígitos.
Impacto	Ejecuta Comando enviado por ACG utilizando Driver. Envía Comando al Driver para responder peticiones de servicios del ACG. Encripta /desencripta información y genera nonce. Conoce Claves de Administrador y Claves de Transmisión.

Tabla 4.5.7.5. – Entrada de LEL "EPP"

Sinónimos	EPP
Noción	Dispositivo que solicita servicios del Driver para que ACG pueda Validar Clave de Administrador, Ingresar Clave de Transmisión. Dispositivo utilizado para Ingresar PIN e Ingresar Dígitos. Entidad que necesita mantener una comunicación confidencial con ACG.
Impacto	Solicita al Decodificador desencriptar el Comando encriptado con la Clave. Ejecuta Comando enviado por ACG utilizando Driver. Envía Comando al Driver para responder peticiones de servicios del ACG. Genera nonce. Conoce Claves de Administrador y Claves de Transmisión.

Tabla 4.5.7.6. – Nueva entrada de LEL "EPP"

4.5.7.2 Análisis de CRC Secundarias

Si se observan las CRC Secundarias del Patrón de Seguridad, se tiene que "*Codificador*" y "*Decodificador*" no existen definidos como términos del LEL correspondiente al modelo de la aplicación ACG+Driver+EPP. Pero el Patrón de Seguridad muestra claramente que son abstracciones necesarias para lograr el objetivo de seguridad confidencialidad, en la comunicación entre Alice y Bob. Como en el nuevo modelo a construir, la "Noción" y los "Impactos" de Alice y Bob están contenidos en las nuevas definiciones para ACG y EPP respectivamente, para lograr que ACG y EPP cumplan con el objetivo de seguridad confidencialidad, es necesario incluir estos dos términos del LEL, que forman parte del Patrón de Seguridad, como términos de LEL del nuevo modelo.

Por otro lado, en el modelo del Patrón de Seguridad, tanto "*Codificador*" como "*Decodificador*" colaboran con "*Algoritmo de Encriptación*", que si bien no surge como una CRC

Secundaria (debido a que el algoritmo de deducción de CRC no barre los impactos de los términos de LEL que se transforman en CRC Secundarias en busca de CRC terciarias) es claro que será una CRC Secundaria a sumar al modelo de diseño, como lo sugiere el modelo estático documentado en la plantilla del Patrón de Seguridad. Por esta razón, es necesario incluir el término de LEL "*Algoritmo de Encriptación*" para construir el nuevo modelo de contexto de la aplicación ACG+Driver+EPP.

Se muestran a continuación los nuevos términos de LEL incluidos para modelar la aplicación: "*Codificador*" en la Tabla 4.5.7.7., "*Decodificador*" en la Tabla 4.5.7.8. y "*Algoritmo de Encriptación*" en la Tabla 4.5.7.9.

Por último, "*Codificador*" y "*Decodificador*" originalmente encriptan/desencriptan Mensaje. En la nueva definición, encriptan/desencriptan Comando.

Sinónimo	Codificador
Noción	Entidad que encripta Comando con una Clave.
Impacto	Conoce Algoritmo de Encriptación.

Tabla 4.5.7.7. – Entrada de LEL "Codificador"

Sinónimo	Decodificador
Noción	Entidad que desencripta un Comando con una Clave
Impacto	Conoce Algoritmo de Encriptación.

Tabla 4.5.7.8. – Entrada de LEL "Decodificador"

Sinónimo	Algoritmo de Encriptación
Noción	Transformación matemática para Encriptar y Desencriptar un mensaje con una Clave.
Impacto	Es utilizado por Codificador y Decodificador.

Tabla 4.5.7.9. – Entrada de LEL "Algoritmo de Encriptación"

El resto de los términos de LEL, que dan nacimiento a las CRC Secundarias se mantienen sin modificaciones en el nuevo modelo.

4.5.7.3 Análisis de Escenarios

El objetivo es estudiar los Escenarios de la Aplicación junto con el/los Escenarios que modelan el Patrón de Seguridad para identificar la presencia de este último en la Aplicación.

En los casos estudiados, se observan dos situaciones.

La primera, es que la presencia del Patrón de Seguridad quede en evidencia en una mención del **Título** del Escenario que modela a dicho artefacto entre los **Episodios** que describen los Escenarios de la Aplicación como se puede ver en [Solinas 2010]. Esto es consecuencia de extender a la construcción de Escenarios, el principio de circularidad y vocabulario mínimo utilizado en la construcción de LEL. Este principio, aplicado a la construcción de LEL, establece que en la descripción de la noción e impactos se debe maximizar el uso de signos definidos en el LEL. Ahora bien, su extensión a la construcción de Escenarios, dice "…*que en la descripción de los episodios de cada Escenario, se debe maximizar el uso de los Títulos definidos en la construcción del conjunto de Escenarios*…". Si un Patrón de Seguridad es modelado en términos de uno o más Escenarios, es de esperar que aplicando este principio, cuando se intente modelar una aplicación con requerimientos de seguridad, con frecuencia encontraremos entre los episodios, una mención directa o indirecta al patrón

de seguridad. Directa en el sentido de que puede estar mencionado textualmente el Título del Escenario que modela al Patrón de Seguridad. Indirecta, en el sentido de que pueden estar sugeridos en el texto los servicios de un Patrón de Seguridad.

La segunda situación, la cual se presenta en este trabajo, es que el requerimiento de cumplir un objetivo de seguridad sea lo suficientemente evidente como para que se identifique anticipadamente al momento de construir los casos de uso. Al momento de construir el diagrama de casos de uso de la aplicación, se descubren numerosos casos de uso que demandan cumplir con un objetivo de confidencialidad en la comunicación entre ACG y EPP. Concretamente, para enviar el Comando al EPP debe estar encriptado. Esto lleva a identificarlo como un caso de uso que colabora intensamente para alcanzar los objetivos de seguridad de la Aplicación.

Por lo dicho arriba y para proceder a analizarlos, se copian aquí, los Escenarios "*Envía Mensaje encriptado con Clave*" (Tabla 4.5.7.10) correspondiente al Patrón de Seguridad GOOCA, descripto en el Capítulo 3; y "*Envía Comando encriptado con Clave*" (Tabla 4.5.7.11) descripto mas arriba.

Título	Envía Mensaje encriptado con Clave
Objetivo	Ocultar el contenido del Mensaje
Contexto	Comunicación establecida entre Alice y Bob.
Recursos	Mensaje, Clave
Actores	Alice, Bob
Episodios	Alice envía Mensaje y Clave a Codificador. Codificador encripta Mensaje utilizando Algoritmo de Encriptación y Clave. Codificador devuelve Mensaje encriptado a Alice. Alice envía Mensaje encriptado a Bob. Bob envía Mensaje encriptado y Clave a Decodificador. Decodificador desencripta Mensaje encriptado utilizando Algoritmo de Encriptación y Clave. Decodificador devuelve Mensaje a Bob. Bob lee Mensaje.

Tabla 4.5.7.10. – Escenario "Envía Mensaje encriptado con Clave"

Título	Envía Comando encriptado con Clave
Objetivo	Ocultar el contenido del Comando
Contexto	Comunicación establecida entre ACG y EPP, Driver ejecutado
Recursos	Comando, Clave
Actores	ACG, EPP
Episodios	ACG encripta Comando con Clave y se lo envía a Driver. Driver envía Comando Encriptado a EPP. EPP desencripta Comando Encriptado con Clave. EPP ejecuta Comando.

Tabla 4.5.7.11. – Escenario "Envía Comando encriptado con Clave"

En esta situación, revisando los atributos (Título, Objetivo, Contexto, Recursos, Actores y Episodios) de ambos Escenarios, y utilizando las conclusiones del análisis de CRC Primarias, se puede inferir lo siguiente:

- El reemplazo de ACG por Alice y de EPP por Bob es directo.
- Comando y Mensaje se los puede considerar sinónimos a los fines de la aplicación.
- La descripción de los Episodios del Patrón de seguridad son más detallados y precisos, incluyendo abstracciones que se tuvieron en cuenta el modelar por primera vez la aplicación. Concretamente, la mención a los términos del LEL Codificador y Decodificador.

Con estas conclusiones se construye el nuevo Escenario que modelará el objetivo de seguridad, ahora en términos de un Patrón de Seguridad. Se puede ver en la Tabla 4.5.7.12.

Título	Envía Comando encriptado con Clave
Objetivo	Ocultar el contenido del Comando
Contexto	Comunicación establecida entre ACG y EPP.
Recursos	Mensaje, Clave
Actores	ACG, EPP
Episodios	ACG envía Comando y Clave a Codificador. Codificador encripta Comando utilizando Algoritmo de Encriptación y Clave. Codificador devuelve Comando encriptado a ACG. ACG envía Comando encriptado a Driver. Driver envía Comando encriptado a EPP. EPP envía Comando encriptado y Clave a Decodificador. Decodificador desencripta Comando encriptado utilizando Algoritmo de Encriptación y Clave. Decodificador devuelve Mensaje a EPP. EPP lee y ejecuta Comando.

Tabla 4.5.7.12. – Nuevo Escenario "Envía Comando encriptado con Clave"

4.5.7.4 Aislamiento del modelo del Patrón de Seguridad

Por qué hablar de aislar el modelo del Patrón de Seguridad..? Para facilitar su identificación y análisis.

Al utilizar la herramienta BMW, se encuentra una única abstracción contenedora vinculada a términos de LEL y Escenarios: "Proyecto". No hay posibilidad de aislar otra entidad, fuera de proyecto pero de manera conjunta con él, como podría ser un sub-escenario. Si bien el conjunto de términos de LEL y Escenarios, al igual que las CRC, pertenecen al Proyecto, cuando se descubre que la dinámica de un Patrón de Seguridad se puede modelar con un Escenario, no hay posibilidad de nombrar esa nueva entidad con la herramienta, con el objetivo de facilitar y estudiar su identificación, análisis y comportamiento, lo cual sería una gran ayuda en esta dinámica iterativa que propone el método.

De los resultados del Análisis de Escenarios se concluyó en dos situaciones que difieren en el modo en que se identifica el Patrón de Seguridad. Ambas conducen al mismo resultado: un Patrón de Seguridad se identifica con el **Título** de un Escenario. Es decir, el Patrón de Seguridad se revela al ingeniero de requerimientos como resultado de alguna de estas dos circunstancias:

- Requerimiento de seguridad no identificado en casos de uso: se menciona de forma directa o evidente el texto del **Título** del **Escenario** que modela el Patrón de Seguridad, entre los **Episodios** que describen alguno de los Escenarios de la Aplicación;

- Requerimiento de seguridad identificado en casos de uso: existe un requerimiento para cumplir un objetivo de seguridad lo suficientemente relevante como para identificarse en el modelo de casos de uso y mapearse directamente con un **Escenario** y por ello a un Patrón de Seguridad.

Para que pueda lograrse esto, es necesario que el Patrón de Seguridad previamente haya sido modelado con LEL y Escenarios, estudiado su dinámica, analizados sus componentes e identificado con el Título de un Escenario a fin de permitir su integración al modelo. Cabe reiterar que esto permite disponer de un conjunto de CRC (las del Patrón de Seguridad) que son de diseño. Para una aplicación con requerimientos de seguridad, no son clases candidatas, son clases impuestas por el diseño del Patrón de Seguridad. Estas CRC conducirán las decisiones a tomar sobre el resto de las CRC, a fin de una integración consistente del Patrón de Seguridad al diseño de la aplicación.

Del análisis de las CRC Primarias y Secundarias, se concluye que los términos del LEL utilizados para modelar el Patrón de Seguridad, son comunes a la aplicación con aquellos requerimientos de seguridad a los cuales satisface el patrón. Podría decirse que era de esperar. Están contenidos en abstracciones que son modeladas con términos propios del dominio de la aplicación, que luego serán asociadas y/o identificadas con abstracciones del Patrón de Seguridad.

El conjunto de Escenarios que modelan la aplicación, en un principio no cambian, salvo que cambien sus requerimientos funcionales.

Como conclusión, el Escenario que representa al Patrón de Seguridad, se modela separado de los Escenarios de la Aplicación, y se lo nombra como Patrón de Seguridad. Tiene los mismos atributos que un Escenario, pero está asilado en otra estructura de datos para su estudio y análisis por separado. Los términos de LEL utilizados para modelarlo, están incluidos en el conjunto de términos de LEL utilizados para modelar la aplicación.

Luego, la heurística implementada en el BMW v1.0, está también implementada para los Patrones de Seguridad. De este modo, es posible modelar un conjunto de Patrones de Seguridad del "núcleo", para su estudio, identificación y análisis en la construcción de modelos de aplicaciones con requerimientos de seguridad, utilizando LEL y Escenarios.

4.5.7.5 Deducción del nuevo conjunto de CRC

Para obtener el nuevo conjunto de CRC de la aplicación con el Patrón de Seguridad embebido en los términos de LEL y Escenarios, se utiliza la nueva versión de la herramienta BMW, la cual permite aislar el Patrón de Seguridad como un Escenario a la vez que permite la integración de sus términos de LEL a los de la Aplicación.

Las CRC primarias deducidas son ACG (Tabla 4.5.7.13); Driver (Tabla 4.5.7.14) y EPP (Tabla 4.5.7.15). Las CRC secundarias deducidas son Codificador (Tabla 4.5.7.16); decodificador (Tabla 4.5.7.17) y Comando, Comandos (Tabla 4.5.7.18).

Primary CRC Card	ACG
Responsabilities	Solicita al Codificador encriptar Comando con la Clave. Envía Comando encriptado al Driver para solicitar servicios del EPP. Genera nonce. Conoce Claves de Transmisión.
Collaborations	*CODIFICADOR, COMANDO, DRIVER, EPP, DECODIFICADOR*

Tabla 4.5.7.13. – CRC primaria "ACG"

Primary CRC Card	Driver
Responsabilities	Recibe Comando enviado por ACG y se lo entrega al EPP. Recibe Comando enviado por EPP y lo entrega a ACG.
Collaborations	*ACG, COMANDO, EPP, CODIFICADOR, DECODIFICADOR*

Tabla 4.5.7.14. – CRC primaria "Driver"

Primary CRC Card	EPP
Responsabilities	Solicita al Decodificador desencriptar el Comando encriptado con la Clave. Ejecuta Comando enviado por ACG utilizando el Driver. Envía Comando al Driver para responder peticiones de servicios del ACG.
Collaborations	*ACG, CODIFICADOR, COMANDO, DECODIFICADOR, DRIVER*

Tabla 4.5.7.15. – CRC primaria "EPP"

Secondary CRC Card	Codificador
Responsabilities	Conoce Algoritmo de Encriptación.
Collaborations	*ACG, EPP, COMANDO, DECODIFICADOR, DRIVER*

Tabla 4.5.7.16. – CRC secundaria "Codificador"

Secondary CRC Card	Decodificador
Responsabilities	Conoce Algoritmo de Encriptación.
Collaborations	*ACG, COMANDO, EPP, CODIFICADOR, DRIVER*

Tabla 4.5.7.17. – CRC secundaria "Decodificador"

Secondary CRC Card	Comando, Comandos
Responsabilities	Es enviado por ACG a Driver para utilizar servicios del EPP. Es enviado por EPP a Driver para comunicarse con ACG.
Collaborations	*ACG, EPP, DRIVER, CODIFICADOR, DECODIFICADOR*

Tabla 4.5.7.18. – CRC secundaria "Comando, Comandos"

Luego, a partir de este conjunto de CRC se puede construir el nuevo diagrama de objetos de la realidad que ahora contiene abstracciones que son de diseño, propuestas por los modelos estáticos y dinámicos del Patrón de Seguridad. De este modo, si se decide utilizar el Patrón de Seguridad en la aplicación, no debería tomarse una decisión que concluya en una inadecuada implementación del Patrón de Seguridad, sugerida por sus modelos: estático y dinámico. Hay otros trabajos de investigación [Heyman 2007] que sugieren controlar la correcta implementación de los Patrones de Seguridad, como una propuesta de métrica de seguridad para la etapa de implementación, lo cual es coherente con lo propuesto en este trabajo.

En la Figura 4.5.7.1 se muestra el diagrama de objetos de la realidad propuesto, a partir de las CRC deducidas. El mismo incluye elementos del Patrón de Seguridad.

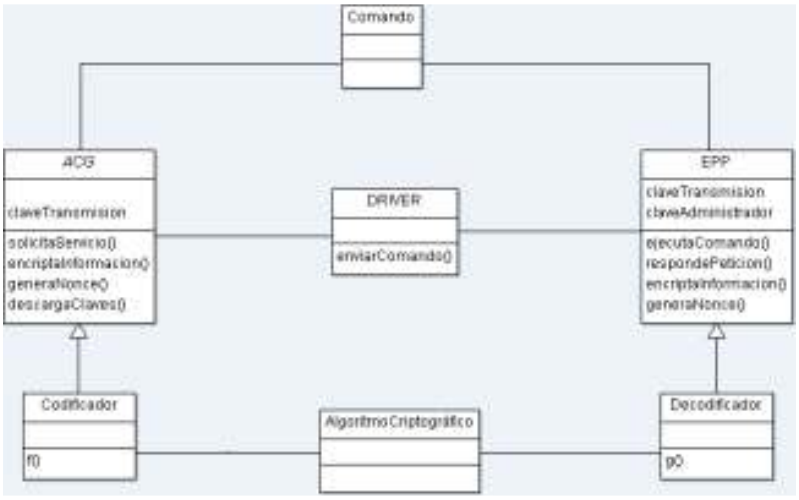

Figura 4.5.7.1. – Diagrama de objetos de la realidad con Patrón de Seguridad

5 Herramienta automatizada

5.1 Importancia de una herramienta automatizada en la construcción de software seguro

Sabemos que existen causas esenciales que producen vulnerabilidades y debilidades en el software [Ozmet 2006]. Una de las más relevantes es la ausencia de tecnología apropiada: herramientas y métodos para asistir el desarrollo y producción de software seguro. Esto ha generado en los últimos diez años múltiples propuestas de metodologías, herramientas y técnicas que enfocan el desarrollo de software seguro, con el objetivo de ayudar a entender y mitigar los problemas de seguridad en las aplicaciones.

Disponer de una herramienta, que permita trabajar con el modelo contextual de los requerimientos de una aplicación con objetivos de seguridad, es un avance significativo. La herramienta debe ayudar a identificar contramedidas para mitigar potenciales amenazas y mapear esas contramedidas sobre Patrones de Seguridad.

Si la misma herramienta permite aislar el Patrón de Seguridad, para estudiar y comprender su funcionamiento con los artefactos del modelo contextual, se está brindando una información adicional para utilizar el Patrón de Seguridad, que hoy no forma parte de la documentación estándar del mismo.

El objetivo final es poder utilizar un Patrón de Seguridad embebido, integrado al diseño de la aplicación. Si después de lo dicho, la herramienta permite integrar el Patrón de Seguridad al modelo de la aplicación y combinar los componentes de ambos en un único resultado, realmente se dispone de un elemento que es determinante el momento de tener que proponer el diseño de una aplicación con requerimientos de seguridad modelados con LEL y Escenarios.

5.2 Baseline Mentor Workbench versión 2.0

Baseline Mentor Workbench ("BMW") es una herramienta que tiene como función asistir al experto del dominio durante la fase de ingeniería de requerimientos, utilizando la metodología del Cient Oriented Requeriments Baseline [Antonelli 1999]. BMW administra las entradas de LEL, Escenarios y tarjetas de CRC.

Dentro del BMW, un Client Oriented Requirements Baseline para un dominio de aplicación dado es llamado proyecto. Dentro de cada proyecto la evolución en el tiempo del mismo es capturada a través de versiones (un proyecto es dividido en varias versiones). Cada versión tiene tres conjuntos: los símbolos de LEL, los Escenarios y las tarjetas CRC. Las entradas del LEL y los Escenarios se deben ingresar manualmente al BMW, sin embargo las tarjetas CRC son generadas automáticamente. Dentro de cada versión las tarjetas CRC son derivadas de las entradas del LEL y los Escenarios pertenecientes a la misma versión. El proceso de derivación es una versión adaptada de la estrategia de derivación presentada en [Leonardi 2001].

Para facilitar el método propuesto en este trabajo, se construyó una nueva versión de esta herramienta, el BMW v2.0.

Esta nueva herramienta, propone separar el Patrón de Seguridad como una entidad nueva con su propio LEL y Escenarios. De modo que ahora BMW también administra Patrones de Seguridad, modelados en términos de LEL y Escenarios.

La arquitectura de la aplicación, que se puede ver en la Figura 5.2.1 es una típica Arquitectura de Repositorio [Sommerville 1995] compuesta por cuatro subsistemas: editor de entradas de LEL, editor de escenarios, editor de patrones de seguridad y generador de tarjetas CRC. En la versión anterior existía un browser de navegación que no fue implementado en la nueva versión.

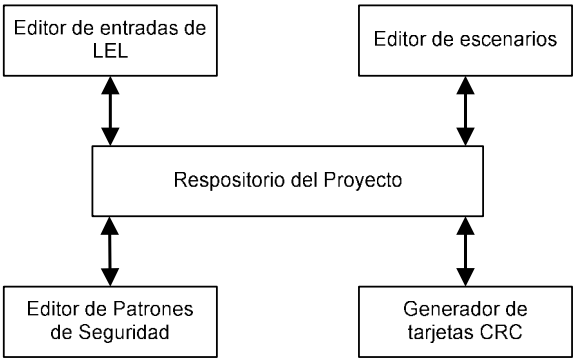

Figura 5.2.1. – Arquitectura de BMW v2.0.

Se intentó mantener las mismas funcionalidades de la versión original, de modo que los editores de entradas del LEL, Escenarios y Patrones de Seguridad proveen ayudas de edición. No se mantuvo la opción de navegación sobre los elementos que componen el modelo.

Con respecto al generador de CRC, BMW v2.0 deriva tarjetas CRC a partir de las entradas del LEL y los Escenarios de acuerdo al algoritmo de derivación expuesto en [Leonardi 2001]. Se suma la opción de utilizar el conjunto de CRC definidos para el proyecto para generar CRC utilizando el Patrón de Seguridad exclusivamente, los Escenarios de la aplicación exclusivamente o ambas cosas simultáneamente. Esto permite tener una visión del conjunto de CRC que se están generando en cada situación y facilitar la aplicación del método propuesto en este trabajo.

Al igual que la versión anterior, la implementación en el BMW v2.0 es una simplificación del algoritmo de derivación expuesto en [Lonardi 2001], que cumple con las etapas principales:

- Hallar clases primarias. Son los actores de los escenarios y/o patrones de seguridad que también están en el conjunto de entradas del LEL.
- Hallar responsabilidades de las clases primarias. Se toman de los impactos de las entradas del LEL.
- Hallar clases secundarias. Son las referencias a entradas del LEL que se encuentran en las responsabilidades de las clases primarias.
- Hallar responsabilidades de las clases secundarias. Se toman de los impactos de las entradas del LEL.
- Hallar colaboraciones. Consiste en determinar con que otras tarjetas CRC participa en la resolución de cada escenario y/o patrón de seguridad.
- Depurar las tarjetas CRC. Consiste en revisar y eliminar tarjetas CRC y atributos repetidos.

La ventana principal, se muestra en la Figura 5.2.2, permite abrir todas las ventanas de la aplicación y acceder a todas las funciones.

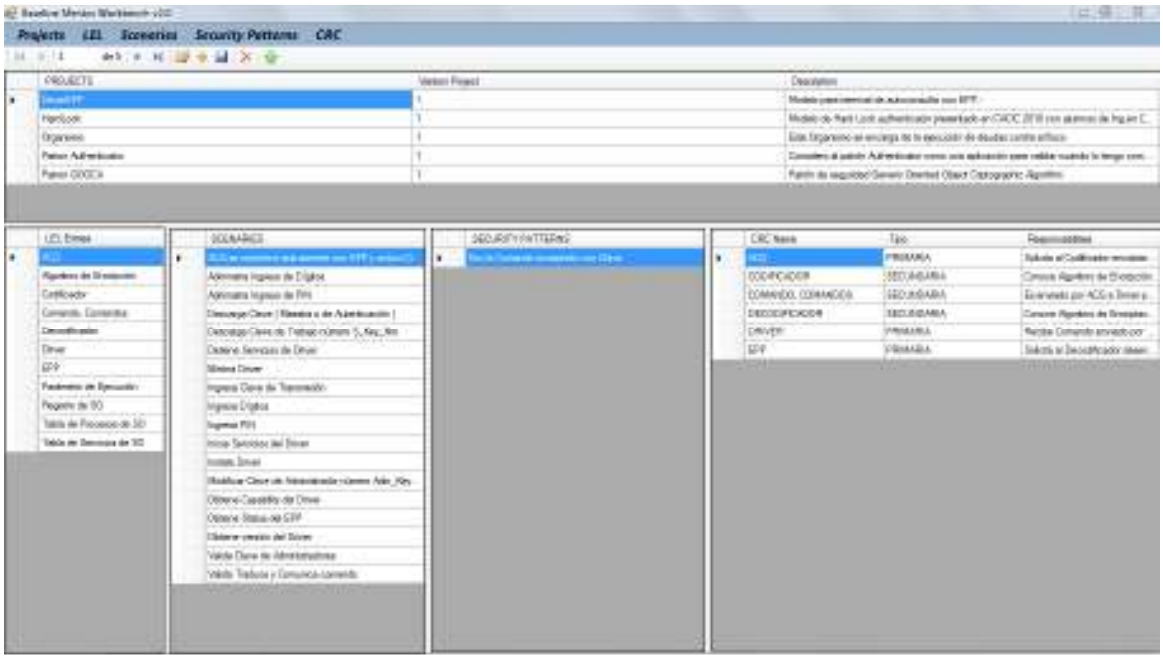

Figura 5.2.2. – Ventana principal del BMW v2.0.

Se intentó mantener la misma apariencia de la versión anterior, presentando una pantalla completa, que muestra los contenidos de LEL, escenarios, patrones de seguridad y CRC a medida que se recorren los proyectos registrados. Luego seleccionando las opciones del menú superior, es posible editar LEL, Escenarios, Patrones de Seguridad o CRC de un proyecto, como puede verse en la Figura 5.2.3.

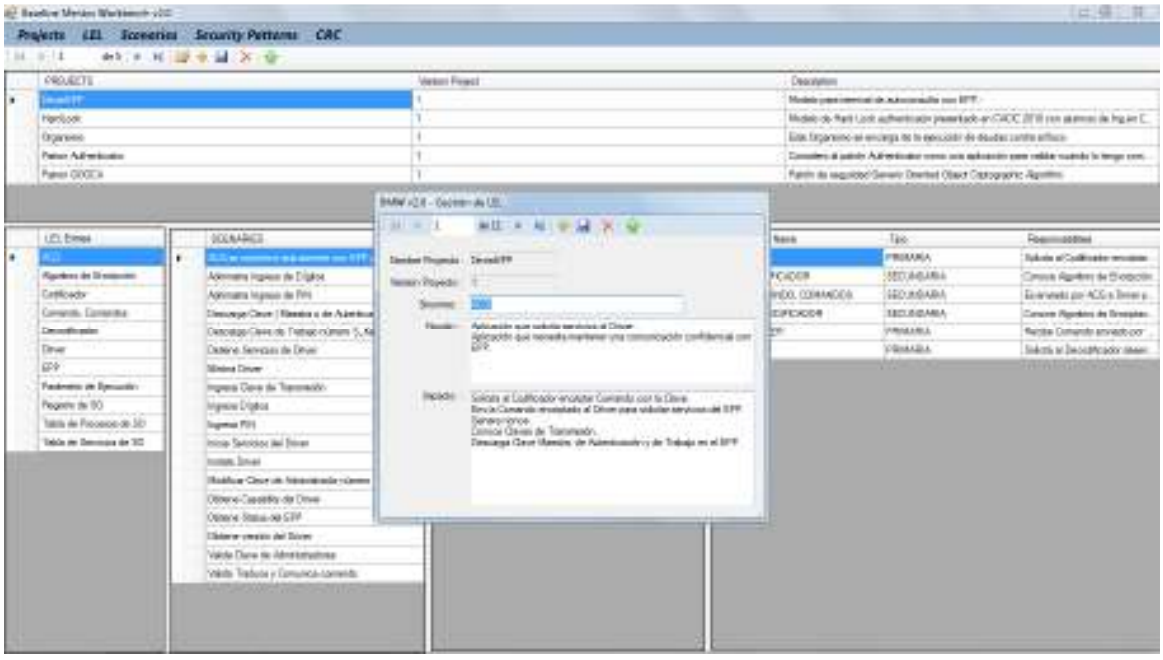

Figura 5.2.3. – Ventana de edición de términos de LEL.

Cada una de estas nuevas ventanas tiene un menú asociado (ver Figura 5.2.4) que se ha mantenido constante para las todas las entidades, incluido Proyecto, LEL, escenarios, patrones de seguridad y CRC.

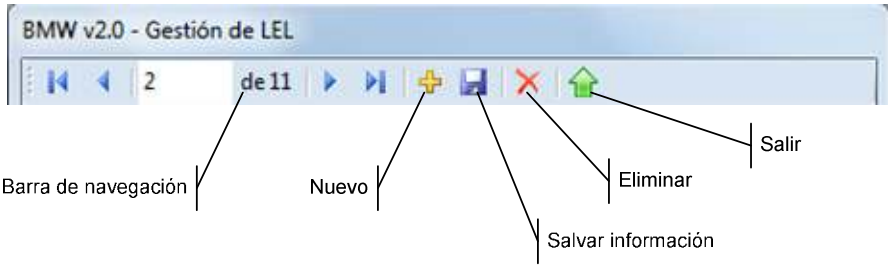

Figura 5.2.4. – Menú asociado común.

La opción "CRC" permite acceder a un menú asociado, ver Figura 5.2.5, que permite editar el conjunto de CRC ya deducidas, deducir las CRC del Patrón de Seguridad exclusivamente con "*Deducir CRC de Security Pattern*" , del proyecto exclusivamente con "*Deducir CRC sin Security Pattern*" ,o de ambos "*Deducir CRC*".

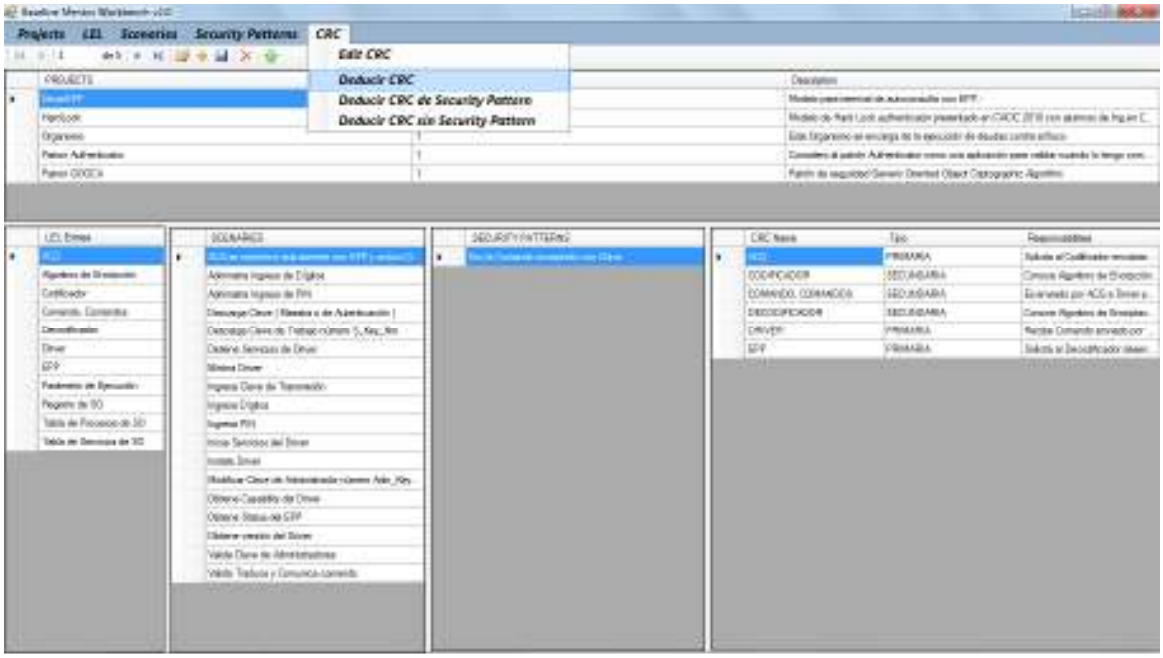

Figura 5.2.5. – Menú asociado CRC.

5.3 Traceability en modelo de LEL, escenarios y patrones de seguridad

"*Forward traceability*" es muy importante para el desempeño de tres funciones [Wieringa 1995]. Al gerente de proyectos le permite estimar el impacto que los cambios de los requerimientos tienen en el diseño y la implementación de software, y viceversa; y descubrir conflictos en forma temprana. Al diseñador le permite desarrollar el producto de software evitando inconsistencias entre sus partes. Y al cliente le permite conocer donde se implementa cada requerimiento.

A estas tres funciones les agrego una más que para el caso de la seguridad considero determinante. Consideremos por un momento el desarrollo de una aplicación en donde los requerimientos de seguridad no han sido tenidos en cuenta desde el inicio, desde la elicitación de requerimientos, análisis de los mismos, diseño e implementación. No hay otra opción, que sumar la

seguridad en etapas posteriores, revisando código, sometiéndolo a rigurosos "*test*" que permitan encontrar aquellos puntos débiles que tendrán que ser sometidos a una modificación o "*refactoring*" seguro con vistas a mejorar su comportamiento frente a ataques. En el mejor de los casos esto se hará previo a su puesta en producción o quizás deban los usuarios y las organizaciones sufrir las consecuencias de una versión que no ha considerado estos aspectos y por lo tanto es vulnerable. Luego serán los errores en el uso, los usos inadecuados o los ataques mal intencionados los que pondrán en evidencia la vulnerabilidad de la aplicación. Estos hechos, al igual que baches en un camino sinuoso poniendo a prueba el sistema de amortiguación del automóvil, pondrán a prueba la débil estructura de seguridad de la aplicación.

Cuando existe una consideración del aspecto seguridad desde etapas tempranas, esos ataques deberían ser contenidos por amortiguaciones que minimicen sus efectos en la funcionalidad de la aplicación. No será posible eliminar el cien por ciento de vulnerabilidades, pero al menos se construirá una estructura para minimizarlas.

Y aquí es donde veo que "*forward traceability*" de requerimientos de seguridad tiene un efecto de brindar la posibilidad de conocer qué estructuras oponen una resistencia a los errores, malos usos o ataques mal intencionados. Al igual que un amortiguador, pensado, calculado y ubicado en determinado lugar dentro de la estructura de un automóvil, los patrones de seguridad, identificados, integrados a la estructura de la aplicación e implementados a partir de considerar requerimientos de seguridad, son una estructura que minimizaran los efectos de errores, malos uso o ataques mal intencionados. No eliminarán completamente la posibilidad de que un atacante encuentre una fisura en la aplicación para obtener beneficios, pero permitirán disminuir la vulnerabilidad de la aplicación. Un amortiguador no elimina por completo la posibilidad que un automóvil sufra daños por un desnivel pronunciado del terreno tomado a excesiva velocidad, pero minimiza sus efectos.

Conociendo la relación entre requerimientos y patrones de seguridad tenemos mayor control y conocimiento de los esfuerzos a los que puede ser sometida la aplicación. Podemos dar ciertas garantías que determinados aspectos han sido tenidos en cuenta. Pensado un error, un mal uso o una ataque mal intencionado exitoso como un esfuerzo sobre la aplicación, que no encuentra su resistencia o amortiguación en un requerimiento que no ha sido tenido en cuenta. Asocio el "tener en cuenta" ese requerimiento a una estructura que se sustancia como Patrón de Seguridad.

El Baseline Mentor Workbench administra LEL, Escenarios, tarjetas CRC e implementa las reglas de Leonardi. Las reglas de "*forward traceability*" fueron motivo de la Tesis de Antonelli. Si bien la propuesta de la herramienta desarrollada en este trabajo es un modelo por etapas, el cual primero se completa el LEL, luego se pasa a construir los escenarios y/o patrones de seguridad, y por último a derivar las tarjetas de CRC, es un modelo que sirve a los objetivos del trabajo: determinar que es posible modelar Patrones de Seguridad utilizando LEL y Escenarios y embeberlos en el modelo de una aplicación con requerimientos de seguridad. Luego, a partir de las reglas de derivación, es posible determinar "*backward traceability*". Este ir y venir entre implementación, Patrones de Seguridad y requerimientos, es lo que permite tener control sobre la estructura que está pensada para soportar los errores, malos usos o ataques mal intencionados.

La traza entre CRC y términos de LEL está determinada por las reglas de derivación propuestas, luego, como el foco de atención del proceso de diseño lo lleva el Patrón de Seguridad, y está identificado como parte de los episodios de un escenario, o como un escenario completo, es posible construir la traza de los elementos de diseño con los requerimientos que le dieron origen. Sería deseable que en un futuro trabajo, esta relación pueda estar automatizada en la herramienta a fin de brindar información de "*traceability*" de Patrones de Seguridad de forma totalmente automática.

En este punto hay que mencionar que es importante el planteamiento hecho por [Braz 2008] en el punto 6.2, incorporando la posibilidad de considerar un conjunto de políticas de seguridad. De este modo, los Patrones de Seguridad, identificados con CRC de análisis y diseño, conduciendo el diseño hacia etapas más avanzadas, podrían quedar asociados a políticas de seguridad a las que dan cumplimiento. De este modo quedaría completa una traza entre amenazas, políticas de seguridad, requerimientos y Patrones de Seguridad. Si se tiene en cuenta que las políticas de seguridad son directivas para toda una organización, sería posible trazar en la construcción de software, el cumplimiento de políticas de seguridad. Esto daría una solidez al desarrollo de software en lo referente a aspectos de seguridad que de otro modo no se podría lograr.

5.4 Futuros trabajos sobre la herramienta

La construcción de esta herramienta cumple con el único objetivo de validar el método propuesto en este trabajo. Se ha utilizado en la construcción de modelos de requerimientos de seguridad para trabajos finales de materias de grado [Solinas 2010] y en un proyecto integrador [Medina 2009], ambos de la carrera de Ing. en Computación de la Facultad de Ciencias Exactas Físicas y Naturales de la Universidad Nacional de Córdoba.

Las observaciones recogidas en estos trabajos, mas la experiencia de utilizar herramientas similares, lleva a la propuesta de realizar un proyecto integrador de la carrera de Ingeniería en Computación para completar las siguientes funcionalidades:

- Construcción de browser de navegación.
- Investigación, determinación y construcción de reglas de "*traceability backward*", incluyendo políticas de seguridad, a fin de determinar en qué políticas de seguridad se han originado los requerimientos que dieron lugar a tal o cual patrón de seguridad.
- Incluir entre los datos del Patrón de Seguridad, información de la plantilla estándar de documentación. Particularmente sus modelos UML estáticos y dinámicos. El objetivo es Modelar los Patrones de Seguridad que forman el núcleo del conjunto de ellos, para tener una base de datos de patrones de seguridad modelados con LEL y Escenarios.
- Incluir en la herramienta la posibilidad de modelar dispositivos de hardware simples, como puede ser un PIC, para mostrar la posibilidad de uso en la construcción de aplicaciones que incluyen dispositivos de hardware como estos.

6 Trabajos relacionados

6.1 Casos de mal uso y casos de abuso

La adaptación de los Casos de Uso ha dado lugar a un conjunto muy prometedor de aproximaciones en el área desarrollo de requerimientos que atacan el problema de la seguridad del software. Para distinguirlos del Caso de Uso estándar, les han asignado varios nombres: casos de abuso, casos de mal uso, casos de uso hostiles, y casos de fiabilidad; estos últimos enfocados en las excepciones. Como elemento en común, intentan ver el software desde el punto de vista de un atacante. A los fines de este trabajo y utilizando el elemento en común, se los identifica a todos ellos como Casos de Mal Uso (CMU) o "*Misuses Cases*", si bien presentan diferencias.

Del mismo modo en que los Casos de Uso se utilizan exitosamente para elicitar requerimientos, los CMU se utilizan para identificar potenciales amenazas. A partir de ellas es posible elicitar requerimientos de seguridad o Casos de Uso de seguridad. La interacción del usuario autorizado con el sistema es diagramada simultáneamente con las interacciones del atacante. Y así como en los Casos de Uso las conexiones entre actor y acción se etiquetan con términos de "extiende" e "incluye", las conexiones en un CMU son etiquetadas con "amenaza" y "mitiga". De este modo, los CMU son el punto de partida para construir un conjunto de Casos de Uso seguros a fin de contrarrestar cada una de las amenazas. Al igual que un Caso de Uso, cada CMU conduce un requerimiento y el correspondiente escenario de prueba para el software. De este modo gran parte del trabajo, invertido en construir los CMU, conduce al desarrollo de requerimientos funcionales de seguridad. No obstante, la técnica de análisis, provee una forma indirecta de elicitar requerimientos no funcionales para proteger el software.

En la construcción de los CMU hay una analogía muy productiva con el siguiente juego: *el mejor movimiento de las blancas consisten en pensar por anticipado la mejor movida de las negras*. Por ejemplo, si el Caso de Uso que se desea analizar es el del robo de un automóvil, como se muestra en la Figura 6.1.1, el jugador de blancas es el propietario legal del vehículo y el jugador de negras el ladrón. La libertad del Conductor estará puesta en riesgo por el Ladrón, de modo que el Conductor necesita "Bloquear el auto". Este es un requerimiento derivado que mitiga la amenaza y ocurre en el nivel más alto de análisis. En el siguiente nivel de análisis se comienza con la respuesta del Ladrón frente a la acción del Conductor de "Bloquear el auto". Si el Ladrón logra acceder al auto y cortocircuitar el arranque del vehículo, está burlando el bloqueo del auto y por ende poniendo bajo amenaza el requerimiento derivado "Bloquear el auto". Para mitigar esta amenaza, se puede tomar la decisión de bloquear el arranque, solicitando el ingreso de una clave para desbloquearlo. Este es un nuevo requerimiento funcional derivado. Se puede observar un movimiento en zig-zag, balanceado entre las decisiones de jugar y contrajugar, propuesto por los usuarios (blancas) y el atacante (negras).

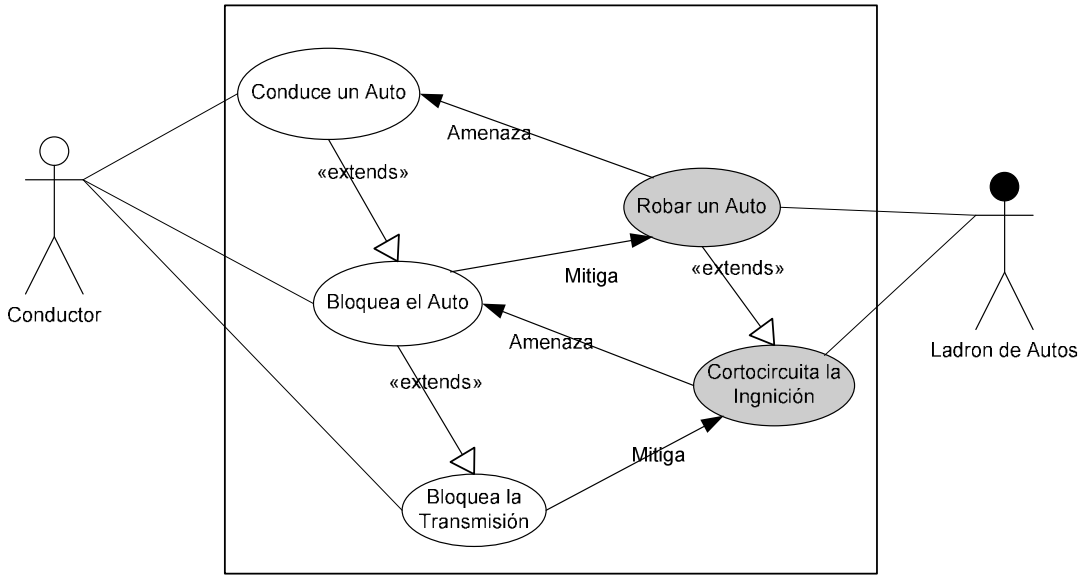

Figura 6.1.1. – Casos de Uso para "Seguridad de Automovil"

Los "requerimientos" surgidos a partir de considerar las decisiones del atacante, disparan extensiones de requerimientos funcionales, que se pueden mapear sobre Patrones de Seguridad. Es una metodología perfectamente compatible con el método propuesto en este trabajo. Puede ser útil un juego de blancas y negras a fin de descubrir en una primera instancia, a partir de estudiar las decisiones de un atacante, un conjunto de potenciales extensiones que luego se mapean sobre Patrones de Seguridad. Luego el método continuaría con normalidad, con la secuencia propuesta en el punto 6.4.

La obra [Hope 2004] es una muy buena introducción al tema. El artículo hace una distinción entre CMU y casos de abuso, observando la "*no intencionalidad*" en la ocurrencia de los primeros y la "*intencionalidad*" en los segundos, y por ello mismo considerados hostiles. Es una distinción similar a la existente entre los riesgos y amenazas, pero no es una distinción estándar. También sugiere utilizar patrones de ataque a fin de ayudar a identificar CMU. Esta tarea debe ser realizada en equipo, donde participen ingenieros de software expertos en el dominio y expertos en el dominio de la seguridad.

Para una descripción concisa de la técnica, se puede consultar [Damodaran 2006]. De esta obra es el siguiente párrafo:

> *Descripción esencial de cómo construir CMU: para CMU, hacer una tormenta de ideas para identificar el modo en que los agentes negativos intentarán impedir o frustrar algunos de los pasos en la descripción del caso de uso; esto conduce a los principales CMU. Durante la sesión, los participantes deben enfocarse en identificar las múltiples formas en que un atacante podría causar daño en el caso de uso bajo estudio; luego podrán completarse otros detalles del ataque. Cada uno de estos modos de ataque es un candidato firme a un CMU.*
>
> *El objetivo es identificar amenazas a la seguridad en cada una de las funciones, áreas, procesos, datos y transacciones involucradas en el caso de uso a partir de potenciales riesgos, como pueden ser los accesos no autorizados desde adentro y desde afuera; ataques de DoS; violaciones a la privacidad, confidencialidad e integridad y ataques malintencionados por parte de hackers. Además de estudiar los modos de ataque, el proceso podría también intentar descubrir posibles errores de*

los usuarios y la correspondiente respuesta de la aplicación. Frecuentemente estos errores podrían causar errores serios en la funcionalidad o en la seguridad de la aplicación. Identificando todas las acciones inapropiadas que podrían ocurrir, capturamos todas las acciones de uso anormal de la aplicación por parte de usuarios genuinos, en términos de accidentes o errores por descuido, o por parte de atacantes intentando quebrar o dañar el correcto funcionamiento de la aplicación.

6.2 Extensión de CMU para la elicitación de requerimientos de seguridad

Fabricio A. Braz , Fernandez Eduardo B. y VanHilst Michael, [Braz 2008] proponen identificar amenazas, partiendo de una extensión del uso de CMU para describir un método sistemático de elicitación de requerimientos de seguridad que luego serán mapeados sobre Patrones de Seguridad. Los autores proponen mirar al sistema desde un nivel de abstracción alto. Por ejemplo, si un atacante se pone como objetivo o meta robar la identidad de un cliente o transferir dinero a su propia cuenta, se debe estudiar el contexto en el que se desarrollan estas actividades a fin de descubrir y definir los requerimientos de seguridad para evitar que se vea comprometido el sistema.

La primera actividad que llevan a cabo es el análisis del flujo de eventos de un caso de uso, o grupo de casos de uso, en el cual cada actividad es analizada con el objetivo de descubrir amenazas relacionadas. Este análisis se realiza sistemáticamente para todos los casos de uso del sistema. La segunda actividad es una selección de políticas de seguridad apropiadas para detener y/o mitigar las amenazas identificadas.

Analizar el flujo de eventos de un caso de uso implica una investigación detallada de cada actividad de cada caso de uso, con el objetivo de tomar conocimiento de cualquier forma posible de subvertirlo, por atacantes internos o externos. Toda la información relacionada a las acciones de los CMU es documentada en un diagrama de actividad UML que ha sido extendido del siguiente modo. Los rectángulos de línea punteada denotan actividades de mal uso o amenazas. Las líneas de conexión punteadas representan flujos de control relacionados al mal uso, las que asocian actividades de mal uso y sus objetivos. En la Figura 6.2.1 se muestra un ejemplo de diagrama de actividad extendido, en el cual se han identificado varias amenazas, nombradas T1 a T8, tanto de fuentes internas como externas, tal cual se muestra en la Figura 6.2.2. Estas amenazas las obtiene el ingeniero de software utilizando su conocimiento de la aplicación. La aproximación es sistemática en el sentido de que se analizan todas las actividades en todos los casos de uso. Por ejemplo, T4 indica que después que un cliente ha provisto información personal, el administrador podría tomar la decisión de difundir ilegalmente esa información a otras personas.

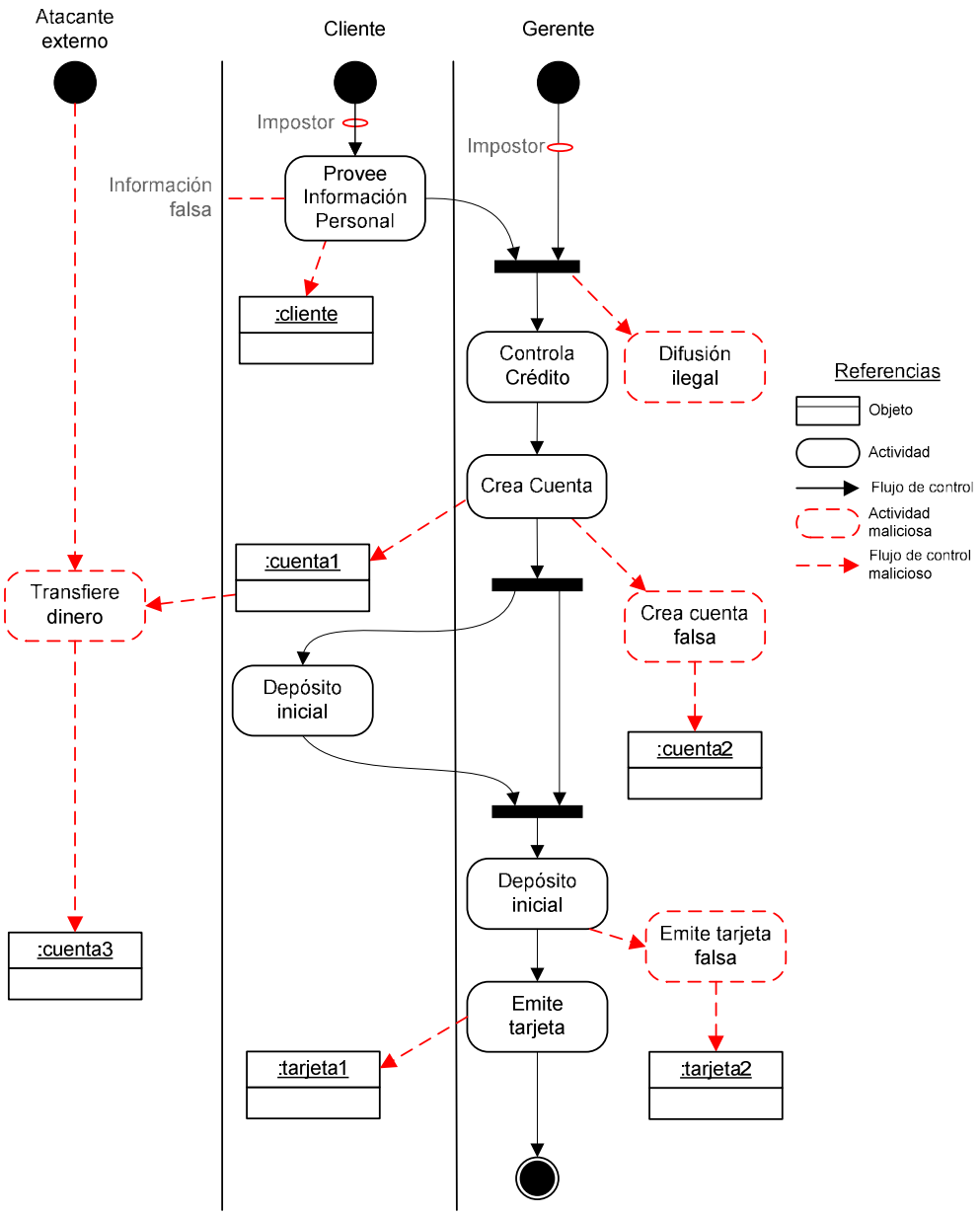

Figura 6.2.1. – Diagrama de actividad extendido (traducción de [Braz 2008])

Una vez que las acciones de mal uso han sido descubiertas, se seleccionan las políticas de seguridad apropiadas para detenerlas y/o mitigarlas. Esta actividad podría ser ayudada por una lista razonable de políticas de seguridad. De todos modos, esta selección debería resultar en un conjunto mínimo de mecanismos a fin de hacer práctico su uso. Por ejemplo, para mitigar las amenazas T5 y T6, como se muestra en la Figura 6.2.2, se puede seleccionar la política de "separación de responsabilidades".

Lista de Amenazas	
T1	El cliente es un impostor. Abre una cuenta y realiza movimientos en nombre de otra persona.
T2	El cliente provee información falsa (financiera, localización) para obtener una cuenta.
T3	El Gerente es un impostor que recoge información ilegalmente.
T4	El Gerente recoge información de los clientes para utilizarla ilegalmente.
T5	El Gerente crea una cuenta falsa con información de los clientes.
T6	El Gerente crea tarjetas falsas para acceder a las cuentas.
T7	Un atacante trata de evitar el acceso de un cliente a sus cuentas (DoS).
T8	Un atacante trata de mover dinero desde una cuenta a su propia cuenta.

Relación para evitarlas o antenuarlas
T3, T1 / P1
T2 / P2
T4 / P3
T6, T5 / P4
T7 / P5
T8 / P6

Lista de Políticas	
P1	Autenticación Mutua
P2	Verificación de fuente de información
P3	Logging
P4	Separación de responsabalidades
P5	Protección contra DoS (redundancia, IDS, etc.)
P6	Autorización

Figura 6.2.2. – Atenuación de amenazas y Políticas de Seguridad (traducción de [Braz 2008]).

Si bien esta aproximación muestra una forma sistemática de identificar y diagramar amenazas y seleccionar las adecuadas políticas, carece de detalles que ayudarían a la aplicación del método. Esto se hace evidente cuando después de aplicar las dos tareas principales propuestas, surgen las siguientes preguntas:

- ¿Cómo realizar el análisis de cada actividad con el fin de recabar todas (o la mayoría) las amenazas?
- ¿Cómo llegar desde la acción mal uso a la política de seguridad apropiada de una manera sencilla?

Las políticas de seguridad de alto nivel de una organización son directivas o guías generales de la seguridad, aplicables a cualquier servicio, práctica o producto, que la organización debe cumplir. Están definidas de acuerdo a las prioridades del negocio, incluyendo su visión, objetivos de comercialización y regulaciones o estándares. Idealmente, toda organización debería tener estas políticas muy bien definidas y claramente comunicadas a sus empleados. Sin ellas, es imposible conocer qué tan segura es la organización, establecer necesidades en esta área y valorar el esfuerzo y dinero a invertir en la tarea de provisión de esta seguridad.

Para descubrir las amenazas asociadas, los autores proponen tener en cuenta tres aspectos cuando se realice este análisis. El primero se refiere a los casos de uso a partir de los cuales se espera encontrar todos los escenarios del sistema. Algunas veces, incluso, es necesario analizar una secuencia de casos de uso los cuales representan un flujograma crítico del negocio, de modo que los diagramas de actividad arriba mencionados podrían mostrar no sólo un caso de uso puntual sino el de un flujograma de negocio. Teniendo en cuenta el ejemplo mencionado en la Figura 6.2.1, una transferencia ilegal de dinero en T8 depende de la creación de una cuenta falsa, de modo que ambas tareas deberían ser analizadas de manera conjunta.

El segundo aspecto a tener en cuenta es el conjunto de atributos de seguridad. Toda actividad del diagrama de actividad debería ser analizada de acuerdo a los principales atributos de seguridad: confidencialidad, integridad, disponibilidad y auditabilidad. Esto significa que debe analizarse si la actividad podría estar en riesgo de comprometer su confidencialidad, integridad, disponibilidad o

auditabilidad. Como tercer aspecto, se debe tener en cuenta la fuente de la amenaza, la cual se relaciona con los privilegios que el agente de amenazas debe tener para ejecutar el ataque. Aquí se puede dividir el conjunto de potenciales atacantes en dos grupos: externos e internos con y sin autorización. Otro aspecto importante a tener en cuenta y no considerado explícitamente, que afecta el análisis del contexto del sistema, está representado por las características del medio ambiente, incluyendo el personal, dinámica de operaciones y otros sistemas en el lugar.

En la Figura 6.2.3, se puede apreciar el resultado de aplicar esta aproximación al diagrama de actividad de la Figura 6.2.1. Esta tabla resume los resultados de aplicar la aproximación de diagramas mal uso a las amenazas encubiertas. La cual es una mejora propuesta por los autores a la representación de la Figura 6.2.2.

		Actividades maliciosas				
Actor	Acción	#	Seg.Att. CO/IN/AV/AC	Source Ain/Uin/Out	Descripción	Bien
Cliente	Provee Info Personal	T1	AC	Ain	Niega haber abierto esta cuenta.	Cuenta
		T2	AV	Out	Destruye aplicación	N/A
		T3	CO	Out/Uin	Realiza espionaje.	Cliente
		T4	CO	AIn	Descubre la relación del cliente con la institución y trata de crear nueva cuenta con esta información	Clliente
		T5	IN	AIn	Provee información falsa (finaciera, localicación).	Cliente
		T6	IN	AIn	Provee información de otra persona (nombre, dirección, DNI).	Cliente
Gerente	Controla Crédito	T7	AC	Ain	Niega haber modificado información de crédito de clientes	Cliente
		T8	CO	AIn	Recoge información personal de los clientes para difundir ilegalmente	Cliente
		T9	CO	Out/UIn	Realiza espionaje.	Cliente
		T10	CO	Out	Recoge datos ilegalmente como un impostor	Cliente
		T11	IN	AIn	Cambia la información de crédito de los clientes para conseguir más clientes	Cliente
Gerente	Crea Cuenta	T12	AC	Ain	Niega la creación de cuentas falsas	Cuenta
		T13	CO	Ain	Recoge información personal de los clientes para difundir ilegalmente	Cuenta
		T14	CO	Out/Uin	Realiza espionaje.	Cuenta
		T15	IN	Ain	Crea una cuenta falsa	Cuenta
Cliente	Depósito onicial	T16	-	-	-	-
Gerente	Emite Tarjeta	T17	IN	Ain	Crea/autoriza tarjeta falsa.	Tarjeta
Cliente	Transfiere dinero	T18	AC	AIn	Niega haber autorizado una transferencia de dinero	Cuenta
		T19	AV	Out	Destruye aplicación	N/A
		T20	CO	Out/Uin	Realiza espionaje.	Cliente
		T21	IN	Out	Transfiere dinero entre cuentas de forma ilegal.	Cuenta

CO - Confidencialidad; IN - Integridad; AV - Disponibilidad; AC - Auditabilidad; Out - Externo; AIn - Interno Autorizado; UIn - Interno No Autorizado

Figura 6.2.3. – Plantilla para análisis de actividades de maliciosas (traducción de [Braz 2008]).

Por último, proponen mapear amenazas con Patrones de Seguridad, asociados mediante políticas de seguridad, como una forma de minimizar la acción de las mismas, como se muestra en la Figura 6.2.4. Se puede consultar el trabajo completo en [Braz 2008].

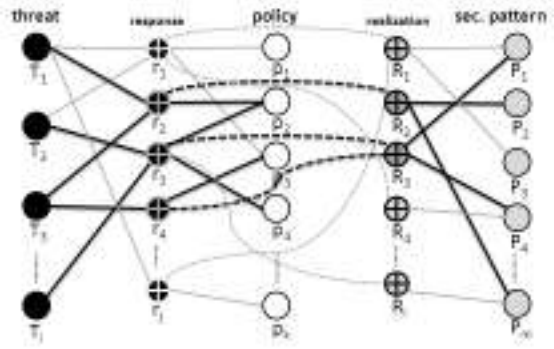

Figura 6.2.4. – Asociación entre amenazas, políticas y patrones de seguridad ([Braz 2008]).

Aquí las políticas de seguridad "p_k" pueden conformar, en general, un conjunto de respuestas "r_j" a las amenazas "T_i". Luego de definida la combinación de respuestas a una amenaza, utilizando políticas de seguridad, es posible sustanciar la respuesta mediante "R_l", la cual puede ser una combinación de patrones de seguridad "P_m".

Este método, al igual que el anterior, se complementa perfectamente con el método propuesto en este trabajo ya que posteriormente a la identificación de "P_m" sería necesario poder integrarlo en un diseño e implementación con los resultados de la elicitación, análisis y diseño de la aplicación propia del dominio que incluye requerimientos de seguridad. Este punto de vista sería de gran ayuda al momento de enfrentar el proceso de construcción de casos de uso de la aplicación, teniendo en cuenta las amenazas y las políticas de seguridad definidas por la organización, para luego abordar la etapa de diseño con la herramienta BMW y los Patrones de Seguridad modelados en términos LEL y Escenarios.

6.3 Árboles de ataque y Casos de Mal Uso

Árbol de ataque, árbol de amenazas y grafo de ataque son representaciones de un posible ataque en contra de un objetivo, en forma de árbol, que ayudan a visualizar una serie de eventos más complejos (patrones de ataque) que pueden ser combinados para comprometer la seguridad de una aplicación. Todos ellos proveen formatos alternativos para capturar información de casos de abuso y casos de mal uso. La visión integral que proveen los árboles y grafos aportan claridad a las dependencias existentes entre patrones de ataque que explotan vulnerabilidades de software y aquellas que tienen como meta vulnerabilidades de otros niveles, como pueden ser la red, el personal o los procedimientos. Este punto de vista puede incrementar el conocimiento para la gestión de riesgos, y mejorar la coordinación de contramedidas para minimizar las vulnerabilidades a través de todas las capas del sistema.

El siguiente ejemplo de árbol de ataque, corresponde al análisis hecho sobre el sitio web de una aerolínea hipotéticamente inseguro.

Objetivo: Hacer una reserva aérea falsa.

1. Persuadir al empleado de sumar una reserva.
   1.3. Chantajear empleado.
   1.4. Amenazar empleado.
2. Acceder y modificar la base de vuelos.
   2.3. Realizar una inyección SQL desde el sitio web (V1).
   2.4. Logearse a la base de datos.
      2.4.1. Adivinar la clave.
      2.4.2. Espiar la clave (V7).

2.4.3. Robar la clave del servidor web (AND).
2.2.3.3. Crear una cuenta en el servidor web.
2.2.3.3.1. Explotar un “buffer overflow” (V2).
2.2.3.3.2. Obtener acceso a una cuenta de empleado.
2.2.3.4. Explotar un “estado de corrida” para acceder a un archivo protegido (V3).

Luego se asocian las vulnerabilidades a los diferentes componentes del sistema: servidor web de la empresa (P), base de datos conteniendo la información sobre reserva de vuelos (Q) y servicios web que administran las cuentas de pasajeros frecuentes (R). Y se agregan las amenazas de que puedan realizarse análisis de tráfico entre los diferentes nodos.

- V1 (nodo P), Inyección SQL. Un usuario autenticado puede introducir una consulta maliciosa que le permitirá leer o modificar cualquier registro de la base de datos.
- V2 (nodo Q) Buffer Overflow. La Common Gateway Interface (CGI) de la página que carga y muestra la información de vuelos, copia el número de vuelo en un pequeño buffer estático sin chequear posibles desbordamientos de buffer.
- V3 (nodo Q). Una condición de ejecución en un comando local le permite al atacante leer cualquier archivo en la máquina donde se encuentra el servidor web.
- V4 (nodo R) Autenticación Débil. El acceso a cada una de las cuentas de viajeros frecuentes es protegido por un PIN de 4 dígitos.
- V5 (enlace entre servidor web y base de datos. Se podría realizar un análisis de tráfico.
- V6 (enlace entre servidor web y servicio de pasajeros frecuentes). Idem V5.
- V7 (sobre script de la base de datos que permite actualizar millaje de pasajeros). Idem V6.

Luego a partir de esta información se podría realizar una evaluación de riesgo ponderando las vulnerabilidades a fin de tomar contramedidas sobre aquellas que representan mayor riesgo. Luego evaluar el impacto de las medidas que se tomen. Ver [Balzarotti 2006].

Este es un ejemplo sencillo. Debido a su tamaño y complejidad, no todos los árboles de ataque se prestan para un análisis manual. Algunos de ellos han sido generados para representar ataques en la realidad, sobre grandes sistemas distribuidos; estos artefactos incluyen cientos y hasta miles de ramas y caminos simultáneos diferentes para completar el ataque.

De acuerdo a quien los intente utilizar, pueden resultar difícil, sino imposibles de utilizar por usuarios no expertos en seguridad. Un árbol es una “lista de precondiciones relacionadas de seguridad”, por lo que resulta poco realista esperar que un no experto en seguridad pueda generar dicha lista. Por esta razón, hay empresas de desarrollo como Microsoft que han descubierto que estas listas de precondiciones relacionadas de seguridad, laboriosamente generadas, en realidad forman patrones que pueden ser estandarizados y reutilizados y que luego son fácilmente comprensibles para desarrolladores no expertos en seguridad. Esta aproximación mediante la utilización de patrones para modelar los ataques y amenazas a la seguridad, han desplazado los modelos que utilizan árboles de ataque, amenazas y grafos.

Si bien los árboles de ataque son comúnmente utilizados para evaluar diseños propuestos, también han sido utilizados para elicitar requerimientos de seguridad a fin de identificar amenazas y contramedidas. [Opdahl 2009] ha hecho un trabajo de comparación entre ambas técnicas, considerando las variables “*efectividad*”, medida en términos del número de amenazas encontradas; “*cobertura*”, medida en términos del tipo de amenazas encontradas; y “*percepción de uso*”, medida en términos de “*utilidad*”, “*facilidad de aplicación*” e “*intencionalidad futura de uso*” y ha llegado a la siguiente conclusión: El árbol de ataque es más efectivo para descubrir amenazas, en particular

cuando se compara con los hallazgos realizados utilizando CMU, sin brindar al usuario una ayuda previa de cómo utilizar esta última técnica (concretamente un ejemplo de CMU). Por otro lado, los usuarios tienen una opinión similar sobre ambas técnicas y la percepción sobre su uso no está influenciada por la performance lograda en el uso de una u otra.

La recomendación de los autores de este último trabajo y a la cual el autor de este informe adhiere, es que ambas técnicas se puede utilizar de manera complementaria a fin de lograr una mejor cobertura y entendimiento de las amenazas y contramedidas a tomar al momento de construir una aplicación con requerimientos de seguridad.

7 Conclusiones

En este trabajo se abordó el estudio de la construcción de software seguro con el objetivo de proponer un método que utilice las mejores prácticas en la resolución de problemas de seguridad; entendiendo que las mejores prácticas en la resolución de problemas de seguridad están presentes en los patrones de diseño, específicos del dominio de la seguridad. Para ello, se consideró que una aplicación está compuesta por el sistema de software propiamente dicho y la infraestructura de comunicaciones que lo soporta. Esta última demanda un tratamiento propio y diferente, que no ha sido abordado en este trabajo. Por otro lado, si el objetivo es construir un sistema de software seguro y considerando que el Ciclo de Vida del Desarrollo del Software (CVDS) tiene múltiples etapas, se concluyó que es determinante la incorporación de la "seguridad" como un aspecto a tener en cuenta en todas y cada una de dichas etapas. Fundamentalmente, se mostró la necesidad de incorporar el aspecto "seguridad", en las primeras etapas del CVDS y el valor que representa esta decisión para las etapas siguientes, las que se ven directamente afectadas en su desarrollo y construcción.

Cuando se habló de Patrones de Seguridad, primero se destacó la fortaleza de trabajar con patrones de diseño en general, ya que sus características son aplicables a los primeros. A saber:

- Aportan una solución bien conocida y por lo tanto muy bien testeada.
- Son claras las ventajas y desventajas de su uso y son conocidas por anticipado de modo que pueden ser tenidas en cuenta al momento del diseño y la implementación.
- Establecen un vocabulario común que facilita la comunicación entre "*stakeholders*".

Estas características logran que la construcción de la "seguridad" se vea facilitada en su estudio, conocimiento y aplicación.

Por otro lado, si bien existe un amplio espectro de Patrones de Seguridad documentados, se dejó claro que los "patrones" que se pretenden utilizar para la construcción de software seguro, forman un núcleo del conjunto y están definidos como tales, con una descripción relevante del problema, una solución intencional, construible en términos de entidades de software, correctamente distribuidos espacialmente y con una configuración adecuada de su comportamiento, el que describe las iteraciones con participantes abstractos externos.

Luego, en el desarrollo de este trabajo se descubrió que estos Patrones de Seguridad son perfectamente descriptos en términos de LEL y Escenarios. Esto permite la construcción de un modelo conceptual de requerimientos del Patrón de Seguridad. Este modelo conceptual es la expresión de una hipotética elicitación de requerimientos, la cual satisface el Patrón de Seguridad, y tiene un valor relevante al momento de integrarlo con el resto de los requerimientos funcionales de un sistema de software. Obviamente, se trata de la construcción de un sistema de software que demanda servicios de seguridad. La información necesaria para la construcción de este modelo conceptual se encuentra en la plantilla de documentación del Patrón de Seguridad.

Como resultado del trabajo de experimentación en la construcción del modelo conceptual de Patrones de Seguridad, se concluyó en casos de éxito en lo que se aplicó el proceso de construcción que se muestra en la Figura 7.2.

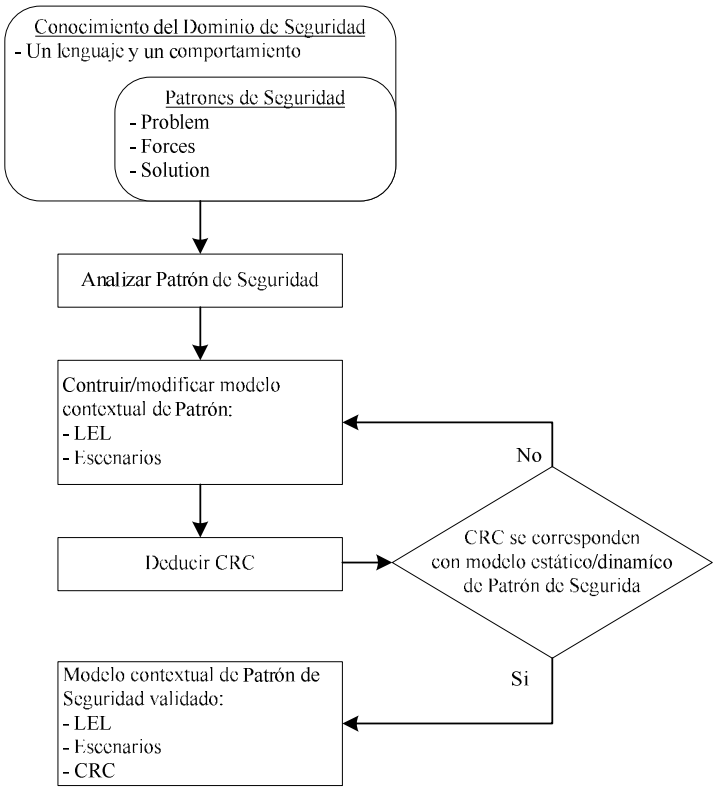

Figura 7.1: Proceso de construcción de un modelo contextual de requerimientos para un Patrón de Seguridad

También se pudo encontrar un mecanismo para la integración de los Patrones de Seguridad con los requerimientos funcionales del sistema de software, el cual se muestra en la Figura 7.2. Esta abstracción del mecanismo se obtuvo "a priori" de la construcción de los modelos conceptuales de diferentes Patrones de Seguridad en términos de LEL y Escenario. El mismo propone una integración natural, no forzada, del dominio de la seguridad, contenido en los Patrones de Seguridad, con el dominio propio del sistema de software que se desea construir, expresado por sus requerimientos funcionales. También modelados conceptualmente, estos últimos, en términos de LEL y Escenarios.

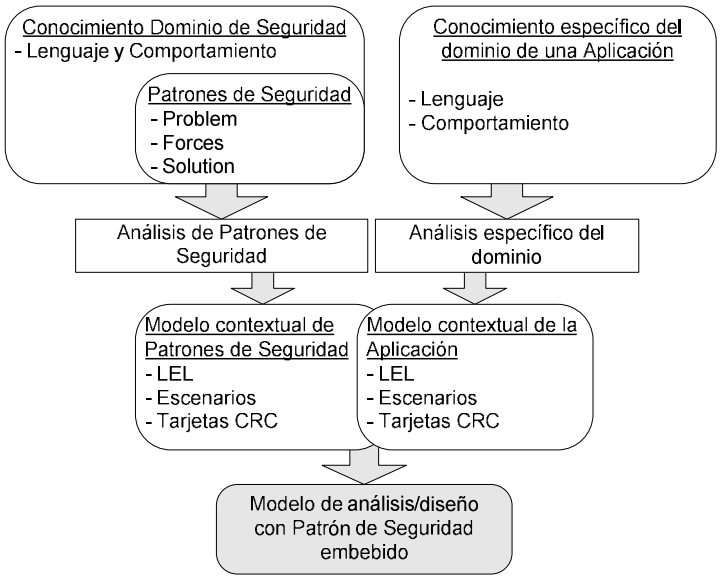

Figura 7.2. – Descripción de Metodología

Esta idea, desarrollada y explorada en la construcción de sistemas de software con requerimientos de seguridad, condujo a encontrar y definir los pasos que constituyen el método definitivo para construir software seguro, que se describe en este trabajo. El mismo propone embeber los Patrones de Seguridad en los requerimientos funcionales de un sistema de software con demanda de servicios de seguridad. Los pasos son los siguientes:

E1: Modelar Patrones de Seguridad.

E2: Modelar Aplicación.

E3: Identificar Patrones de Seguridad.

E4: Analizar CRC Primarias.

E5: Analizar CRC secundarias.

E6: Analizar Escenarios.

E7: Deducir CRC.

E8: Reutilizar Código.

Utilizando este método se construyó la aplicación descripta en el Anexo I, la que se encuentra en producción en el mercado financiero de terminales de auto-consulta. También se aplicó el método en la construcción de sistemas conformados por una parte de software y otra parte de hardware, como un "*hardlock authenticator*", el cual fue publicado como un caso de éxito de aplicación en la Ingeniería en Computación.

De este modo, se cumple el objetivo expresado en la propuesta inicial de Trabajo de Tesis, que planteaba como posible describir un método para la construcción de sistemas de software seguro, que se inicie en el conocimiento contenido en los artefactos documentados en la plantilla del Patrón de Seguridad. Este conocimiento hace posible encontrar un modelo conceptual de requerimientos. Estos

requerimientos son en definitiva las amenazas que el patrón de seguridad anula o limita, y el Patrón de Seguridad representa la mejor práctica para resolver el problema.

Por otro lado, la construcción del modelo conceptual del Patrón de Seguridad es un camino hacia atrás en el CVDS. Se inicia en el análisis del texto contenido en los títulos "*problem*", "*forces*" y "*solution*", de la plantilla de documentación del Patrón de Seguridad y va en búsqueda de un conjunto de CRC que luego son validadas con las clases que modelan estáticamente al Patrón de Seguridad. De este modo, el conocimiento y el comportamiento del micro dominio que representa el Patrón de Seguridad, expresado en términos de LEL y Escenarios, ahora "es" el Patrón de Seguridad.

También se muestra que una vez asociado el Patrón de Seguridad con un conjunto de LEL y Escenarios, estos elementos permiten su identificación, directamente en la semántica asociada a sus requerimientos funcionales, presente en los casos de uso del sistema de software con requerimientos de seguridad o entre los episodios que los describen como escenarios.

Disponer del Patrón de Seguridad en estos términos, permite la integración natural con el modelo conceptual de los requerimientos funcionales de un sistema de software, obviamente, también expresado en términos de LEL y Escenarios. Se aplica la misma heurística a ambos dominios integrados, y el Patrón de Seguridad es un timón que orienta las decisiones para un diseño que contemple las mejores prácticas en la solución de problemas de seguridad.

Así se cumple con el objetivo de describir un método de construcción de software seguro que aplique el conocimiento adquirido sobre el problema de elicitación de requerimientos de seguridad y posterior modelo de análisis, utilizando LEL, Escenarios y Patrones de Seguridad.

Cabe destacar la importancia de la decisión de proponer un método que sume los requerimientos de seguridad como parte de los requerimientos funcionales del sistema de software. De este modo no existe una instancia separada en donde sean tenidos en cuenta, a posteriori de la construcción de un modelo de diseño o implementación de los requerimientos funcionales del sistema de software que se pretende construir. Sino que forman parte del conocimiento y comportamiento que se tiene de sus requerimientos funcionales.

Para resumir y concluir, el método descripto en este trabajo, muestra que es posible modelar Patrones de Seguridad utilizando LEL y Escenarios. Luego es posible embeber el modelo contextual del dominio de una aplicación con requerimientos de seguridad, con el modelo contextual del Patrón de Seguridad.

No hay que olvidar que en todo momento se utiliza lenguaje natural, lo cual facilita el estudio, análisis, identificación y aplicación de los Patrones de Seguridad. Esta es una ventaja de la utilización de LEL y Escenarios para la construcción de los modelos conceptuales de requerimientos.

Tomada la decisión de utilizar un Patrón de Seguridad y embeberlo en la aplicación, la misma cumple una función de "conducir" las decisiones referentes a la construcción del software en la dirección de las etapas posteriores del CVDS: diseño e implementación. Esto garantiza que las mejores prácticas para solucionar aspectos de seguridad serán utilizadas en la construcción del sistema de software. Y como se hace de manera consistente, utilizando elementos comunes para los modelos contextuales del dominio de la aplicación y del Patrón de Seguridad, el método ataca las inconsistencias que puedan aparecer entre los requerimientos de unos y otros.

Por otro lado, aunque no se haya estudiado ningún caso, dada la analogía entre casos de uso y casos de mal uso, se pueden complementar ambos métodos para identificar amenazas, identificar Patrones de Seguridad y modelar requerimientos de aplicaciones seguras teniendo en cuenta las políticas de seguridad de una organización.

Así como se utiliza STRIDE/DREAD para identificar bienes a proteger, es posible utilizar árbol de amenazas y luego tomar la decisión de modelar requerimientos para identificar Patrones de Seguridad y continuar con la aplicación del método propuesto en este trabajo.

Se debe realizar un estudio más exhaustivo del núcleo de Patrones de Seguridad factibles de ser utilizados como artefactos de software, para modelarlos con LEL y Escenarios y utilizarlos en aplicaciones con requerimientos de seguridad. Esto permitiría dar una mayor cobertura a las actividades de reconocimiento y selección de Patrones de Seguridad y evaluar estadísticamente el método propuesto, meta que supera los objetivos propuestos inicialmente para este trabajo. Junto a esta tarea, sería deseable contar con este "núcleo" de Patrones de Seguridad, incorporados como modelos conceptuales en la herramienta BMWv2.0, conjuntamente con los elementos de la plantilla de documentación de cada patrón y el código que pudiera disponerse de cada uno. Esto completaría una herramienta útil para el diseño de software seguro.

Por último, aquellos que utilicen LEL y Escenarios para modelar requerimientos, se verán beneficiados con esta propuesta ya que dispondrán de un método para incorporar Patrones de Seguridad en sus modelos construidos con un método que ha demostrado [Leonardi 2001][Antonelli 2003] ser efectivo para el estudio de otros dominios.

8 Futuros trabajos

El siguiente paso es completar la herramienta para que permita navegar los requerimientos, incluidos los Patrones de Seguridad, lo que mejoraría las posibilidades de "*traceability*". Por otro lado, debería recopilarse el núcleo de Patrones de Seguridad en su totalidad. En un principio esos patrones tendrían código asociado para su implementación. Modelarlos con LEL y Escenarios y luego sumarlos a la herramienta para que pasen a formar parte de su base de datos, para su estudio y utilización en el modelo de aplicaciones con requerimientos de seguridad. De este modo se tendría una herramienta con el núcleo de Patrones de Seguridad como punto de partida para la elicitación de requerimientos de aplicaciones con demanda de cumplimiento de políticas de seguridad.

Hay que profundizar el estudio del el impacto que tiene la trazabilidad de requerimientos en la generación de software seguro. Si bien se ha podido ver la relación entre amenazas, políticas de seguridad y patrones de seguridad propuesta por Braz [Braz 2008], y los conceptos expuestos arriba en 6.2, es importante poder contar con resultados de experiencias que permitan validar nuevas hipótesis al respecto.

Hay que identificar una métrica asociada a los Patrones de Seguridad, para poder evaluar su utilización en etapas posteriores al diseño. Thomas Heyman y Christophe Huygens [Heyman 2008] de Catholic University of Leuven, Bélgica, están investigando una métrica asociada directamente con Patrones de Seguridad, a fin de medir su efectividad para asegurar el software. Se pueden encontrar los aspectos comunes con el trabajo de Heyman para implementar la métrica con la herramienta aquí descripta. De este modo, se dispondrá de una herramienta para diseñar software seguro, que ya no tan solo utiliza las mejores prácticas para dar cumplimiento a los objetivos de seguridad especificados en la etapa de requerimientos, sino que también podrá brindar métricas para valorar el cumplimiento de esos objetivos y por ende la seguridad del software.

Un trabajo inmediato es estudiar y documentar casos reales, como el presentado en este trabajo, de aplicación del método, conjuntamente con Casos de Mal Uso o su extensión propuesta por Braz y Fernández en [Braz 2008]. Esto permitiría integrar el método con otras herramientas y métodos, también utilizados en la etapa de elicitación de requerimientos de seguridad.

Es indispensable atacar el problema de construir aplicaciones que utilicen Patrones de Seguridad, de forma autónoma. Que identifiquen de forma automática los Patrones de Seguridad a partir de modelos contextuales de requerimientos expresados en lenguaje natural. Hay que abordar esta problemática con un modelo que permita evolucionar requerimientos hacia una aplicación más segura, mutando estructuras alrededor de Patrones de Seguridad, de forma totalmente automática. El modelo a seguir es el de los seres vivos, con su creatividad biológica, mutando nuevos individuos, y por fin encontrando una propuesta más evolucionada, mejor adaptada; para el caso más seguras. Así como se estudia la evolución de cúmulos de galaxias reducidas a puntos en un modelo, para inferir comportamientos del universo, se podría estudiar la evolución de estos modelos contextuales con Patrones de Seguridad, con el objetivo de evaluar los mejores modos de utilizarlos en aplicaciones con requerimientos de seguridad. El modelo propuesto por Gregory Chaitin en [Chaitin 2011] es esperanzador, intenta establecer una serie de reglas o leyes generales básicas que deberían cumplir los desarrollos de software evolutivo. Así como él plantea la vida como software en evolución, yo digo que en los próximos años empezaremos a hablar de software como vida evolucionando en supercomputadoras en la dirección del cumplimiento de los requerimientos que demandan nuevas aplicaciones, cada vez más complejas y con mayores problemas de seguridad.

9 Referencias

[Ambysoft 2006] Scott W. Ambler, (Toronto, ON, Canada: Ambysoft, Inc., December 15, 2006). Disponible en: http://www.ambysoft.com/unifiedprocess/ ; Accedido en Agosto 2011.-

[Ammala 2000] Darwin Ammala, "A New Application of CONOPS in Security Requirements Engineering," CrossTalk: The Journal of Defense Software Engineering (August 2000).

[Amysoft 2006-1] Scott W. Ambler, The Agile Unified Process (Toronto, ON, Canada: Ambysoft, Inc., June 12, 2006). Disponible en: http://www.ambysoft.com/unifiedprocess/agileUP.html ; Accedido en Agosto 2011.-

[Alexander 2002] Ian Alexander, Modelling the Interplay of Conflicting Goals With Use and Misuse Cases, in Proceedings of 8th International Workshop on Requirements Engineering Foundation for Software Quality, Essen, Germany, September 9–10, 2002: 145–152.

[Antonelli 2003] Antonelli L.,"Traceability en la elicitación y especificación de requerimientos"; Facultad de Informática – Universidad Nacional de La Plata, Argentina, 2003.

[Arisholm 1999] Erik Arisholm and Dag I.K. Sjøberg (University of Oslo), and Jon Skandsen and Knut Sagli (Genera AS), "Improving an Evolutionary Development Process—a Case Study" (paper presented at European Software Process Improvement (EuropSPI99), Pori, Finland, October 25–27, 1999). Disponible en: http://www.iscn.at/select_newspaper/process-models/genera.html ; Accedido en Agosto 2011.-

[Balzarotti 2006] Balzarotti Davide (Milan Polytechnic), Monda Mattia (University of Milan), and Sicari Sabrina (University of Catania), "Assessing the Risk of Using Vulnerable Components," chap. in Quality of Protection: Security Measurements and Metrics, Dieter Gollmann, Fabio Massacci, and Artsiom Yautsiukhin eds. (New York, NY: Springer, 2006). Disponible en: http://homes.dico.unimi.it/~monga/lib/qop.pdf ; Accedido en Agosto 2011.-

[Basin 2006] D.A. Basin, J. Doser, T. Lodderstedt, Model driven security: from UML models to access control infrastructures, ACM Transactions on Software Engineering and Methodology 15 (1) (2006) 39–91.-

[BDR98] Alexandre M. Braga, Ricardo Dahab, and Cecilia M.F.Rubira. PayPerClick: Um Framework para Venda e Distribuicao On-line de Publicasoes Baseado em Micropagamentos. In SBRC'98 - 16o Simpósio Brasileiro de Redes de Computadores, page 767, Rio de Janeiro, RJ, Brasil, May 1998. Extended Summary.-

[Beznosov 2006] Konstantin Beznosov "Towards Agile Security Assurance," presentation given at the Calgary Agile Methods User Group (CAMUG), Calgary, Alberta, Canada, University of Calgary, 3 October, 2006.-

[Boehm 81] Barry W. Boehm, *Software Engineering Economics* (Upper Saddle River, NJ: Prentice-Hall, 1981).-

[Boehm 1984] Barry W. Boehm, et al., "A Software Development Environment for Improving Productivity," IEEE Computer 17, no. 6 (June 1984): 30–44.

[Bra00] A. Braga, C. Rubira, and R. Dahab, "Tropyc: A pattern language for cryptographic object-oriented software", Chapter 16 in Pattern Languages of Program Design 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.). Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.53&rep=rep1&type=pdf; Accedido en Agosto 2011.-

[Braz 2008] Fabricio A. Braz , Fernandez Eduardo B. y VanHilst Michael, "Eliciting Security Requirements through Misuse Activities"; DEXA '08 Proceedings of the 2008 19th International Conference on Database and Expert Systems Application; IEEE Computer Society Washington, DC, USA ©2008.-

[Bus 96] Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture, Volume 1. A System of Patterns, Wiley, New York, NY, 1996.- [CGHK98] Pau-Chen Cheng, Juan A.Garay, Amir Herzbertg, and Hugo Krawczyk. A Security Architecture for the Internet Protocol. IBM Systems Journal, 37(1):42-60, 1998.-

[Chaitin 2011] Graitin, Gregory; "Life as Evolving Software"; Disponible en: http://www.cs.umaine.edu/~chaitin/; Accedido en Agosto 2011.-

[CLASP 2005] "The CLASP Application Security Process" Secure Software, Inc. Copyright (c) 2005, Secure Software, Inc.-

[Cop 2011] James O. Coplien. A pattern definition. Disponible en: http://www.hillside.net/patterns/222-design-pattern-definition; Accedido en Agosto 2011.-

[CORAS 2011] Disponible em: http://coras.sourceforge.net/; Accedido en Agosto 2011.-

[Damodaran 2006] 200 Meledath Damodaran, "Secure Software Development Using Use Cases and Misuse Cases," Issues in Information Systems VII, no. 1 (2006): 150–154. Disponible en: http://www.iacis.org/iis/2006_iis/PDFs/Damodaran.pdf; Accedido en Agosto 2011.-

[Das98] F. Das Neves and A. Garrido, "Bodyguard", Chapter 13 in Pattern Languages of Program Design 3, Addison-Wesley 1998.-

[EH 2003] Ben Elsinga, Aaldert Hofman. "Security taxonomy pattern language". In The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003), Germany, 2003.

[Ess97] W. Essmayr, G. Pernul, and A.M. Tjoa, "Access controls by object-oriented concepts", Proc. of 11th IFIP WG 11.3 Working Conf. on Database Security, August 1997.-

[Eup 2011] Scott W. Ambler, "Enterprise Unified Process (EUP)" [home page] (Toronto, ON, Canada: Ambysoft, Inc.,March 1, 2006).Disponible en: http://www.enterpriseunifiedprocess.com/; Accedido en Agosto 2011.-

[evo 2011] "Evolutionary Delivery" [web page] (Bilthoven, The Netherlands: N.R. Malotaux Consultancy). Disponible en: http://www.malotaux.nl/?id=evo; Accedido en Agosto 2011.-

[Fer06c] E.B.Fernandez, T. Sorgente, and M.M. Larrondo-Petrie, "Even more patterns for secure operating systems', Procs. of the Pattern Languages of Programming Conference (PLoP 2006).-

[Fer93] E.B.Fernandez, M.M.Larrondo-Petrie and E.Gudes, "A method-based authorization model for objectoriented databases", Proc. of the OOPSLA 1993 Workshop on Security in Object-oriented Systems , 70-79.-

[Fer94] - E. B. Fernandez, J. Wu, and M. H. Fernandez, ``User group structures in object-oriented databases", Proc. 8th Annual IFIP W.G.11.3 Working Conference on Database Security, Bad Salzdetfurth, Germany, August 1994.-

[Fernandez 2006] E. B. Fernandez, M.M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, "A methodology to develop secure systems using patterns", Chapter 5 in "Integrating security and software engineering: Advances and future vision", H. Mouratidis and P. Giorgini (Eds.), IDEA Press, 2006, 107-126.

[Fernandez 2007] Eduardo B. Fernandez; Hironori Washizaki; Nobukazu Yoshioka2; Atsuto Kubo; and Yoshiaki Fukazawa; "Classifying Security Patterns".-

[Fernandez 2007_1] Fernandez E.B., Larrondo-Petrie M.M., Sorgente T., VanHilst M., "A Methodology to Develop Secure Systems Using Patterns"; 2007.-

[FISMA 2011] Federal Information Security Management Act of 2002; Disponible en: http://en.wikipedia.org/wiki/Federal_Information_Security_Management_Act_of_2002; Accedido en Agosto 2011.

[Forsberg 1991] Kevin Forsberg and Harold Mooz, “The Relationship of System Engineering to the Project Cycle,” in Proceedings of the First Annual Symposium of National Council on System Engineering, October 1991: 57–65.

[Fowler 1999] Fowler, M.; "Refactoring: Improving the Design of Existing Code". Addison-Wesley. 1999.

[Garvin 1984] Garvin, D., “What does product quality really mean?”, Sloan Management Review, Vol 26, No 1, 1984.

[Gilliam 2004] David P. Gilliam (NASA Jet Propulsion Laboratory), “Security Risks: Management and Mitigation in the Software Life Cycle,” in Proceedings of the Thirteenth IEEE International Workshop on Enabling Technologies, June 14-16, 2004: 211–216.

[Haley 2004] Charles B. Haley, Robin C. Laney, and Bashar Nuseibeh (The Open University), “Deriving Security Requirements From Crosscutting Threat Descriptions,” in Proceedings of the Third International Conference on Aspect-Oriented Software Development, Lancaster, UK, March 22–26, 2004: 112–121. Available from: http://mcs.open.ac.uk/cbh46/papers/AOSD04.pdf

[Haley 2006] Charles B. Haley, Jonathan D. Moffett, Robin Laney, and Bashar Nuseibeh (The Open University), “A Framework for Security Requirements Engineering,” in Proceedings of the Second Software Engineering for Secure Systems Workshop, Shanghai, China, May 20–21, 2006: 5–42. Available from: http://mcs.open.ac.uk/cbh46/papers/Haley-SESS06-p35.pdf

[Her97] Michael Herfert. Security-Enhanced Mailing List. IEEE Network, pages 30-33, 1997.-

[Heyman 2007] Heyman, Thomas; Huygens, Christophe; Joosen, Wouter; “Developing secure applications with metrics in mind”, Second Workshop on Security Metrics, MetriCon 2.0, Boston, MA, United States, August 7, 2007.

[Heyman 2008] Thomas Heyman, Riccardo Scandariato, Christophe Huygens, Wouter Joosen, “Using Security Patterns to Combine Security Metrics”, Proceedings of the The Third International Conference on Availability, Reliability and Security, ARES 2008 (SecSE Workshop), 2008.-

[HIPAA 2011] The Health Insurance Portability and Accountability Act of 1996 (HIPAA) Privacy and Security Rules. Disponible en: http://www.hhs.gov/ocr/privacy/; Accedido en Agosto 2011.-

[HN98] Amir Herzberg and Dalit Naor. Surf'N'Sign: Client Signatures on Web Documents. IBM Systems Jornal, 37(1):61-71, 1998.-

[Hope 2004] Paco Hope and Gary McGraw (Cigital, Inc.), and Annie I. Antón (North Carolina State University), “Misuse and Abuse Cases: Getting Past the Positive,” IEEE Security and Privacy (May-June 2004): 32–34. Disponible en: http://www.cigital.com/papers/download/bsi2-misuse.pdf; Accedido en Agosto 2011.-

[HY97] Amir Herzberg and Hilik Yochai. Minipay: Charging per Click on the Web. Computer Network and ISDN Systems, 1997.-

[IBM 2011] IBM Corp., “Introduction to business security patterns”, white paper, Disponible en: http://www-03.ibm.com/security/patterns/intro.pdf; Accedido en Agosto 2011.-

[JAV 2011] Sun Developer Network, Disponible en: http://java.sun.com/blueprints/patterns/; Accedido en Agosto 2011.-

[KBZ 2001] Saluka R. Kodituwakku, Peter Bertok, and Liping Zhao. Aplrac: A pattern language for designing and implementing role-based access control. In European Conference on Pattern Languages of Programs (EuroPLoP 2001), 2001. Pattern source.

[Kerievsky 2005] Keirevsky, Joshua; "Refactoring to Patterns"; Addison-Wesley; 2005.

[Lam71] B.W. Lampson, “Protection”, Procs. 5th Annual Conf. on Info. Sciences and Systs.,1971, 437-443. Reprinted in ACM Operating Systs. Review, 8, 1 (January 1974), 18-24.-

[Lamsweerde 2003] Lamsweerde A. v.; Brohez S.,; Landtsheer, R. D.; Janssens, D.; "From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering," in Proceedings of the Requirements for High Assurance Workshop, Monterey Bay, CA, September 8, 2003, 49–56. Disponible en: http://www.info.ucl.ac.be/Research/Publication/2003/avl-RHAS03.pdf; Accedido en Agosto 2011.-

[Leite 1997] Leite, J.C., Rossi, G., et al.: "Enhancing a Requirements Baseline with Scenarios". Proceedings of RE 97, IEEE Third International Requirements Engineering Symposium, IEEE Computer Society Press, 1997, pp 44-53.-

[Leonardi 2001] Leonardi, C., Leite J.C., Rossi G., "Una Estrategia de Modelado Conceptual de Objetos basada en Modelos de Requisitos en Lenguaje Natural", Tesis de maestría, Facultad de informática, Universidad Nacional de La Plata, Argentina, Noviembre, 2001.

[McDermott 2001] John J. McDermott (NRL CHACS), "Abuse–Case-Based Assurance Arguments," in Proceedings of the 17th Annual Computer Security Applications Conference, New Orleans, LA, December 10–14, 2001: 366–374.

[McDermott 2005] John McDermott (Naval Research Laboratory Center for High Assurance Computer Systems), "Attack-Potential-based Survivability Modeling for High-Consequence Systems" in Proceedings of the Third International Information Assurance Workshop, March 2005, 119–130. Disponible en: http://www.nrl.navy.mil/chacs/pubs/04-1226-3437.pdf ; Accedido en Agosto 2011.-

[McGraw 2005] Gary E. McGraw (Cigital Inc.), Risk Management Framework (RMF) (Washington, DC: US CERT, September 21, 2005). Disponible en: https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/risk/250.html; Accedido en Agosto 2011.-

[McGregor & Sykes 2001] McGregor, J., & Sykes, D. (2001). A practical guide to testing object-oriented software. Addison-Wesley.-

[Mead 2006] Nancy R. Mead (CMU SEI), Requirements Elicitation Introduction (Washington, DC: US CERT, September 22, 2006). Disponible en https://buildsecurityin.uscert.gov/daisy/bsi/articles/best-practices/requirements/533.html ; y ibid., Requirements Elicitation Case Studies Using IBIS, JAD, and ARM (Washington, DC: US CERT, September 22, 2006). En: https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements/532.html; Accedido en Agosto 2011.-

[Medina 2009] Medina, Gastón; "Construcción de Driver de usuario para dispositivo Encrypted PINPad utilizando Desarrollo Conducido por Modelos", Proyecto Integrador carrera de Ing.en Computación; Facultad de Ciencias Exactas Físicas y Naturales; UNC; Córdoba 2009.-

[Mei 1999] Mei C. Yatoco (University of Missouri-St. Louis), Joint Application Design/Development (fall 1999). Disponible en: http://www.umsl.edu/~sauter/analysis/JAD.html; Accedido en Agosto 2011.-

[Microsoft SRM 2006] Microsoft Corporation, "Understanding the Security Risk Management Discipline" revised May 31, 2006, chap. 3 in Securing Windows 2000 Server (Redmond, WA: Microsoft Corporation, November 17, 2004. Disponible en: http://technet.microsoft.com/en-us/library/cc751213.aspx ; Accedido en Agosto 2011.-

[Morana 2006] Marco M. Morana (Foundstone Professional Services), "Building Security into the Software Life Cycle: a Business Case" (paper presented at BlackHat USA, Las Vegas, NV, August 2–3, 2006).-

[Mouratidis 2007] H. Mouratidis and P. Giorgini," Integrating Security and Software Engineering: an Introduction," chap. I in Integrating Security and Software Engineering, H. Mouratidis and P. Giorgini, eds. (Hershey, PA: Idea Group Publishing, 2007).

[Maruyama 2007] Maruyama Katsuhisa; "SECURE REFACTORING Improving the Security Level of Existing Code"; in Proc.Int'l Conf.Software and Data Technologies (ICSOFT 2007), pp.222-229, 2007.

[MSD 2011] Microsoft patterns and practices development center, Disponible en: http://msdn.microsoft.com/en-us/practices/default ; Accedido en Agosto 2011.-

[NDIA 2010] National Defense Industrial Association (NDIA) Systems Engineering Division, Top Software Engineering Issues Within Department of Defense and Defense Industry, final vers. 5a. (NDIA, September 9, 2010). Disponible en http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Studies/NDIA%20Top%20SW%20Issues%202010%20Report%20v5a%20final.pdf; Accedido en Agosto 2011.-

[Opdahl 2009] Opdahl Andreas L., Sindre Guttorm; "Experimental comparison of attack trees and misuse cases for security threat identification"; Information and Software Technology Volume 51, Issue 5, May 2009, Pages 916-932, SPECIAL ISSUE: Model-Driven Development for Secure Information Systems.-

[rad 2011] Walter Maner (Bowling Green State University), Rapid Application Development (Bowling Green, OH:Bowling Green State University Computer Science Dept., March 15, 1997). Disponible en: http://csweb.cs.bgsu.edu/maner/domains/RAD.htm; Accedido en Agosto 2011.-

[Rashid 2003] A. Rashid, A.M.D. Moreira, and J. Araújo, "Modularisation and Composition of Aspectual Requirements" in Proceedings of the Second International Conference on Aspect-Oriented Software Development, Boston, MA, March 17–21, 2003: 11–20; and A. Rashid, A.M.D. Moreira, P. Sawyer, and J. Araújo, "Early Aspects: a Model for Aspect-Oriented Requirement Engineering," in Proceedings of the IEEE Joint International Conference on Requirements Engineering, Essen, Germany, September 9–13, 2002: 199–202.

[RUP 2011] IBM/Rational, "Rational Unified Process Best Practices for Software Development Teams" Rational Software White Paper, no. TP026B Rev 11/012001 (November 2001). Disponible en: http://www.ibm.com/developerworks/rational/library/253.html ; Accedido en Agosto 2011.-

[Scandariato 2008] Koen Yskout, Thomas Heyman, Riccardo Scandatiato, Wouter Joosen; "Security patterns: 10 years later", Katholieke Universiteit Leuven, department of Computer Science; 2008.-

[Sch06] M. Schumacher, E.B.Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, Security Patterns, J. Wiley & Sons, 2006.-

[Schumacher 2006] Schumacher M., Fernandez, E. B., Hybertson, D., Buschmann, F., and Sommerlad, P., Security Patterns: Integrating security and systems engineering", Wiley 2006.

[Seacord 2008] Seacord, Robert C. The CERT C Secure Coding Standard. Boston: Addison-Wesley, 2008.-

[sec 2011] The Security Patterns page, maintained by M. Schumacher; Disponible en: http://www.securitypatterns.org; Accedido en Agosto 2011.-

[SFBH+06] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. Security Patterns. Wiley & Sons, 2006. pattern source.

[Smith and Thober 2006] Smith, S. F. and Thober, M. (2006). Refactoring programs to secure information flows. In Proc. PLAS'06, pages 75–84.

[SOAR 2007] Agile Methods: Issues for Secure Software Development; State-of-the-Art Software Security Assurance; Report Information Assurance Technology Analysis Center (IATAC); Data and Analysis Center for Software (DACS); Joint endeavor by IATAC with DACS; July 31, 2007

[Softare Assurance CBK 2007] "Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire and Sustain Secure Software"; Software Assurance Worrkforce Education and Training Working Group; Homeland Security; USA; October 2007.-

[SOx 2011] Sarbanes–Oxley Act; Disponible en: http://en.wikipedia.org/wiki/Sarbanes%E2%80%93 Oxley_Act; Accedido en Agosto 2011.

[Solinas 2009] Solinas Miguel, Fernandez Eduardo B., Antonelli Leandro; "Embedding Security Patterns into a Domain Model"; Procs. of the 3rd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'09), in conjunction with the 6th International Conference on Trust, Privacy & Security in Digital Busines(TrustBus'09), pp. 176-180.-

[Solinas 2010] Solinas Miguel, Trad Jairo, Abdala Juan, Capdevila Francisco, Fernandez Eduardo B., Antonelli Leandro; "Caso de éxito de método que aplica patrones de seguridad en la Ingeniería en Computación"; CACIC 2010; Ciudad de Buenos Aires, 2010.-

[Ste05] C. Steel, R. Nagappan, and R. Lai, Core Security Patterns: Best Strategies for J2EE, Web Services, and Identity Management, Prentice Hall, Upper Saddle River, New Jersey, 2005.-

[Sun 1996] Sun Microsystems, "Workspace Hierarchy Strategies for Software Development and Release," chap. in Sun WorkShop TeamWare 2.0 Solutions Guide, and "Concurrent Development," sec. in Guide (Mountain View, CA: Sun Microsystems, Inc., 1996). Disponible en: http://www.informatik.uni-hamburg.de/RZ/software/SUNWspro/teamware/solutions_guide/hierarchy.doc.html; Accedido en Agosto 2011.-

[UMLSec] http://inky.cs.tu-dortmund.de/main2/jj/index.html

[Viega 2001] John Viega and Gary McGraw, Building Secure Software: How to Avoid Security Problems the Right Way (Boston, MA: Addison-Wesley, 2001).-

[Wilander 2005] Wilander, J.; Gustavsson, J.; (Linköpings University); "Security Requirements: a Field Study of Current Practice" in Proceedings of the Third Symposium on Requirements Engineering for Information Security, Paris, France, August 29, 2005. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.7480&rep=rep1&type=pdf Accedido en Agosto 2011.-

[Winston 1970] Winston W.R.; "Managing the Development of Large Software Systems, Concepts and Techniques" in Proceedings of IEEE Wescon, Los Angeles, CA, 1970 August 1–9, 1970. Disiponible en: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf; Accedido en Agosto 2011.-

[YB 97] Joseph Yoder and Jeffrey Barcalow. Architectural patterns for enabling application security. In The 4th Conference on Patterns Language of Programming (PLoP 1997), 1997. Pattern source.

[Yod97] Yoder, J.; Barcalow, J.; "Architectural patterns for enabling application security". Procs. PLOP'97, Also Chapter 15 in Pattern Languages of Program Design, vol. 4 (N. Harrison, B. Foote, and H. Rohnert, Eds.), Addison-Wesley, 2000. Disponible en: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.3950&rep=rep1&type=pdf ; Accedido en Agosto 2011.

10 Anexo I: Descripción de EPP (Encrypted Pin Pad)

El EPP es un dispositivo de encriptación multipropósito. El modelo que se utilizara en el trabajo es el KY3688/B de la firma Shenzhen KeYu CO., LTD. Este dispositivo combina un teclado de alta durabilidad, capaz de soportar vandalismos, y un módulo de seguridad. El modulo de seguridad provee servicios de encriptación de acuerdo a los estándares estándares “VM PCI certification”, “CE”, “IP64” y “China Banking system certification”.

Figura 10.1 – Apariencia exterior de EPP.

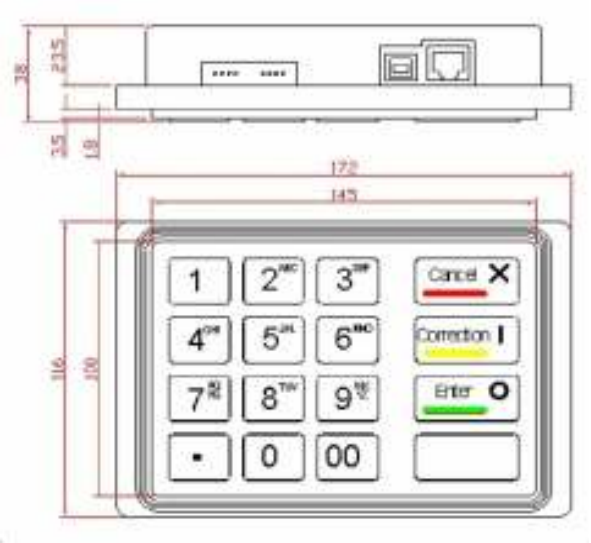

Figura 10.2. – Dibujo de dimensiones de EPP.

La seguridad que posee implementada en acuerdo con la organización del Banco de las Personas de China son:

Servicio de Seguridad	*Norma*
DES	SO8731-1 (igual a ANSI X3.92)
3DES	ANSI X9.52
PIN	ISO9564-1/2 (igual a ANSI X9.8) e IBM3624
¡Error! No se encuentra el origen de la referencia.	ISO8732 (igual a ANSI X9.9)
Seguridad general	PCI

Tabla 10.1. – Normas de Seguridad de EPP.

10.1 Administración de claves del EPP

El proceso de administración de claves se encarga del almacenamiento, descarga y validación de claves en el dispositivo EPP. El usuario Administrador, es el usuario con mayor nivel de privilegios, y es el único autorizado para ejecutar cualquier proceso de administración de claves en el dispositivo EPP. Para la ejecución del resto de las funcionalidades existe el usuario sin privilegios.

10.1.1 Descripción de claves

El dispositivo EPP utiliza cinco tipos de claves para administrar los servicios de seguridad. Las claves están organizadas jerárquicamente como se muestra en la Figura 10.3. Cada clave es utilizada con un único propósito. Todas ellas se almacenan en el EPP y se ingresan utilizando su teclado o son descargadas al EPP por la Aplicación Cliente Genérica. El uso de estas claves permite el Control de Autorización.

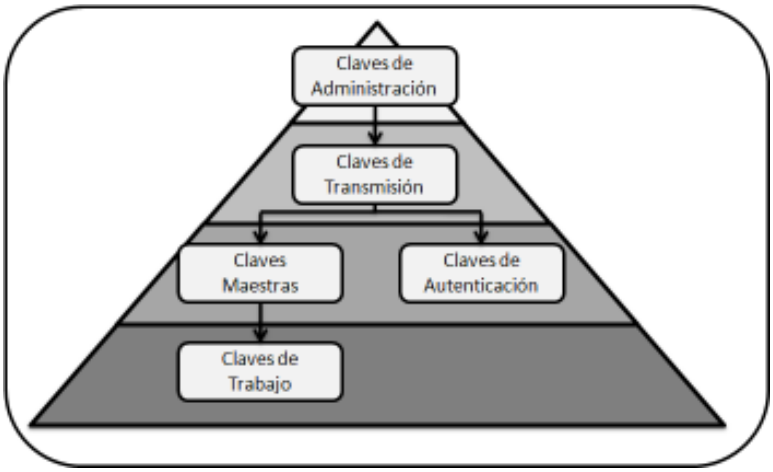

Figura 10.3. – Jerarquía de Claves EPP

Clave de Administrador Adm_Key

El dispositivo EPP almacena dos claves de administrador. Se ingresan por el teclado del EPP. Cada una tiene una longitud de 8 bytes (64 bits). Se utilizan para validar la identidad de dos administradores que pueden ser la misma persona.

Clave de Transmisión Tra_Key

Cada administrador tiene asociada una clave de transmisión de 24 bytes (192 bits) que se almacena en el EPP. Se ingresan por el teclado del EPP. Se utilizan para encriptar/desencriptar las claves maestras M_Key y de autenticación A_Key, cuando éstas son bajadas al EPP por la ACG.

Claves de Autenticación A_Key

Existen 16 claves de autenticación, cada una de las cuales tiene 24 bytes (192 bits). Se utilizan para encriptar/desencriptar el número al azar utilizado en el proceso de autenticación mutua (entre EPP y ACG) y activación de Clave de Trabajo.

Claves Maestra M_Key

Existen 16 claves maestras, cada una de las cuales tiene 24 bytes (192 bits). Cada una de estas claves está asociada a una clave de autenticación y a una clave de trabajo. Se utilizan para encriptar/desencriptar la correspondiente clave de trabajo cuando es enviada por la ACG al EPP.

Clave de Trabajo S_Key

Existen 16 claves de trabajo, cada una de las cuales tiene una longitud de 24 bytes (192 bits). Es utilizada para encriptar/desencriptar datos. Las 16 claves se utilizan para:

- *0 a la 3:* para algoritmo de PIN.
- *4 a la 7:* para algoritmo de MAC.
- *8 a la 11:* para encriptación de datos.
- *12 a la 15:* como clave de encriptación individual.

10.1.2 Funcionalidades específicas para la administración de claves

10.1.2.1 Ingreso de Claves de Administrador

El proceso de ingreso de las dos claves de administrador lo inicia la ACG mediante el envío de un comando al EPP. No existe la posibilidad de ingresar sólo una clave. El EPP anuncia que ingresa en este modo dando un sonido. A continuación, utilizando el teclado numérico del EPP, se debe ingresar la primer clave de Administrador. Si el proceso de proceso de ingreso es correcto, el EPP indica con otro sonido (beep) el inicio del proceso de ingreso de la segunda clave de Administrador. Nuevamente, si el proceso de ingreso de esta segunda clave es correcto, se dice que el proceso de ingreso de claves de administrador ha finalizado exitosamente.

10.1.2.2 Modificación Clave de Administrador

El proceso de modificación de cualquiera de las dos claves de Administrador, al igual que el ingreso, lo inicia la ACG mediante el envío de un comando al EPP. El EPP anuncia que ingresa en este modo emitiendo un sonido. A continuación se debe ingresar la clave de Administrador que se desea modificar. Si el proceso de validación es exitoso, el EPP solicita el ingreso de la nueva clave de Administrador por primera vez. Finalizado el ingreso, correctamente, solicitará que se ingrese la nueva

clave por segunda vez. Si el proceso de ingreso de la nueva clave de administrador finaliza exitosamente la nueva clave será salvada.

10.1.2.3 Ingreso de claves de Transmisión

Primero se debe ejecutar de manera exitosa el proceso "Ingreso de claves de Administrador" (Control de Autorización). A continuación, la ACG inicia la secuencia de ingreso de una clave de transmisión, enviando el comando correspondiente. Al igual que las claves de Administrador, no existe posibilidad de ingresar sólo una. El EPP indica que ha entrado en modo de ingreso de clave de transmisión emitiendo un beep. A continuación, el primer Administrador debe ingresar su parte de la clave de transmisión. Finalizado el ingreso, el EPP indica con otro beep el inicio del proceso de ingreso de la segunda parte de la clave correspondiente al segundo Administrador. Finalizando el proceso se almacena la clave en el EPP.

10.1.2.4 Descarga de clave Maestra – clave de Autenticación – clave de Trabajo

Las claves de Autenticación, Maestras y de Trabajo se descargan y almacenan en el EPP mediante comandos enviados por la ACG. Para realizar la descarga de una clave de Autenticación o de una clave Maestra es necesario tener la autorización de los administradores, para lo cual se debe ejecutar el procedimiento de ingreso de Claves de Administrador (Control de Autorización). Para realizar la descarga (activación) de una clave de trabajo se requiere previamente una autenticación mutua entre ACG y EPP (Autenticación de Operaciones). Luego de que la ACG envía el valor de la clave junto con el índice de clave asociado y la clase a la cual la clave pertenece, el Epp emite un sonido (beep) para indicar que el proceso ha finalizado exitosamente y la clave se ha almacenado.

10.1.2.5 Ingresar PIN y Dígitos

Tanto el proceso de ingreso de PIN y como el proceso de ingreso de dígitos, se encuentran protegidas por un servicio de seguridad que implementa el EPP. Este servicio se conoce con el nombre de "Autenticación mutua y activación de clave de trabajo".

10.1.2.6 Autenticación mutua y activación de clave de trabajo

El proceso de autenticación mutua entre y EPP se inspira en un trabajo propuesto por Needham, R. y Schroeder, M.. Este es un patrón de seguridad conocido como patrón Authenticator. En la Figura 10.4 se puede ver la representación de los pasos del proceso. En el punto de partida, A y B han intercambiado exitosamente sus claves públicas K_{Ua} y K_{Ub}. A continuación, A envía su Identificación ID_A y un nonce N1, ambos encriptadas con la clave pública de B, K_{Ub}. B los recibe, desencripta la información con su clave privada, K_{Rb} y extrae N1. Genera su propio nonce N2, lo adjunta al enviado por A, encripta ambos utilizando la clave pública de A, K_{Ua} y envía el resultado. A recibe esa información, la desencripta con su clave privada, K_{Ra} y extrae N2 y se lo reenvía a B, encriptado con la clave pública de B, K_{Ub}. Llegado a ese punto, ambos extremos han podido demostrar su identidad. El resto es simplemente enviar una clave secreta, Ks para ser utilizada en un servicio de criptografía simétrica. A le envía Ks, asegurando servicios de confidencialidad y autenticación utilizando la clave pública de B, K_{Ub} y su clave privada K_{Ra}.

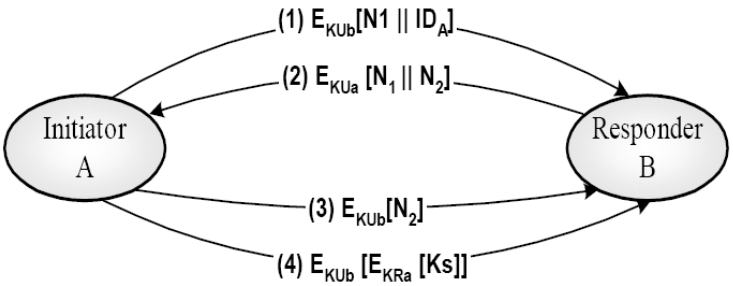

Figura 10.4. – Esquema de Autenticación Needham/Schroeder

Para el caso que involucra la comunicación entre Host y EPP, debido a su implementación no existen los pares de clave pública/privada; en su lugar el Administrador previamente se encarga de descargar en Host y EPP la clave de autenticación número <A_Key_No>, que se corresponde con la clave de trabajo que se desea activar. En el dispositivo EPP el patrón se encuentra implementado a nivel de Hardware. La Figura 10.5, muestra los pasos del proceso que se describe más abajo.

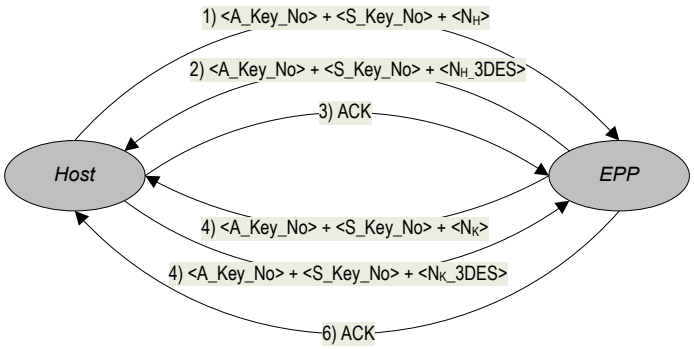

Figura 10.5. - Esquema de Autenticación Host/EPP

Términos utilizados en este proceso:

- **A_Key_No:** índice indicando número de clave de autenticación
- **S_Key_No:** número de clave de trabajo.
- **N_H**: nonce generado por Host para autenticarse con EPP
- **N_H_3des: N_H** encriptado utilizando algoritmo 3DES
- **N_K:** nonce generado por EPP para autenticarse con Host
- **N_K_3des: N_K** encriptado utilizando algoritmo 3DES

Secuencia de ejecución:

1. Host envía comando de petición de autenticación al EPP, con la siguiente información:

   <A_Key_No> + <S_Key_No> + <N_H>
2. EPP recibe petición de autenticación del Host.

Utiliza la clave de autenticación indicada por <A_Key_No> para encriptar con 3DES el valor < N_H >; y envía este autenticador al Host, más <A_Key_No> y <S_Key_No>

$N_H\_3des = 3DES_{\mathbf{K}<A\_Key\_No>}(< N_H >)$

Si ocurriese que la posición indicada por <A_Key_No> está vacía, el comando no podrá ser ejecutado.

3. El Host recibe autenticador enviado por EPP

   Utiliza la clave de autenticación indicada por <A_Key_No> para desencriptar <N_H_3des> y obtener N'_H.

   Compara < N_H > con < N'_H >. Si son iguales el proceso de autenticación de EPP ha sido exitoso y se lo comunica al EPP.

4. EPP recibe resultado de su proceso de autenticación y envía petición de autenticación al Host.

   Si el proceso de autenticación de EPP, iniciado por el Host ha sido exitoso, EPP envía su comando de petición de autenticación al Host, con la siguiente información:

   <A_Key_No> + <S_Key_No> + <N_K>

5. Host recibe petición de autenticación del EPP.

   Utiliza la clave de autenticación indicada por <A_Key_No> para encriptar con 3DES el valor < N_K >; y envía este autenticador al EPP, más <A_Key_No> y <S_Key_No>.

   $N_K\_3des = 3DES_{\mathbf{K}<A\_Key\_No>}(< N_K >)$

   Si ocurriese que la posición indicada por <A_Key_No> está vacía, el comando no podrá ser ejecutado.

6. EPP recibe autenticador enviado por Host.

   Utiliza la clave de autenticación indicada por <A_Key_No> para desencriptar <N_K_3des> y obtener N'_K:

   Compara < N'_K > con < N_K >. Si son iguales, el proceso de autenticación de Host ha sido exitoso y se lo comunica al Host (Autenticación de Operaciones).

10.1.3 Proceso de ingreso de PIN

El proceso de ingreso de PIN requiere de la Autenticación Segura de Ambos Extremos y de la Activación de la clave de trabajo que va a utilizarse durante el proceso antes de habilitarse el dispositivo EPP para el ingreso de un PIN (Autenticación de Operaciones). Las claves de trabajo que se utilizan para el ingreso del PIN son las claves 0, 1, 2 o 3.

Luego de que la Activación de clave trabajo sea exitosa, se inicia el modo de ingreso de PIN. Este proceso consta de 2 etapas:

1. *Recepción de dígitos ocultos:*

   A partir de la habilitación del ingreso de PIN en el EPP, el usuario ingresa cada uno de los dígitos que conforman su número personal de identificación. El EPP envía por cada uno de estos dígitos un digito oculto (‘*’ – 0x2A) que representa a cada uno de estos dígitos ingresados por el usuario.

2. *Recepción de pinblock:*

   Una vez finalizado el ingreso del PIN del usuario, ya sea por llegar al máximo de longitud del PIN o por haber ingresado la tecla ENTER cumpliendo con el mínimo de longitud del PIN, el EPP envía un pinblock de 8 bytes (64 bits) con el PIN completo ingresado padeado a izquierda con 0xFF, es decir que el resto de los bytes que no correspondan al pin serán 0xFF.

El proceso puede ser cancelado presionando la tecla CANCEL, pero si se presiona la tecla CORREGIR, el PIN es reseteado y debe comenzar a ingresarse desde su primer digito.

Todos los datos transferidos en este proceso se encuentran encriptados en 3DES por la clave de trabajo activada inicialmente.

10.1.4 Proceso de ingreso de dígitos

El proceso de ingreso de dígitos también requiere de la Autenticación Segura de Ambos Extremos y de la Activación de la clave de trabajo que va a utilizarse durante el proceso antes de habilitarse el dispositivo EPP para el ingreso de un digito (Autenticación de Operaciones). Las claves de trabajo que se utilizan para el ingreso del PIN son las claves 12, 13, 14 o 15.

Luego de que la Activación de clave trabajo sea exitosa, se inicia el modo de ingreso de digito. Este proceso consta de 1 sola etapa que se ejecuta continuamente:

1. *Recepción de dígitos:*

   A partir de la habilitación del ingreso de dígitos en el EPP, el usuario ingresa un digito. El EPP envía este digito en un bloque de 8 bytes (64bits) padeado a izquierda con 0xFF, es decir que el resto de los bytes que no correspondan al digito serán 0xFF.

El proceso finaliza cuando se presiona la tecla CANCEL o la tecla ENTER. La tecla CORREGIR en este proceso no cumple ninguna funcionalidad pero si es reconocida y si se la presiona el EPP envía el código de tecla 0x08.

Todos los datos transferidos en este proceso se encuentran encriptados en 3DES por la clave de trabajo activada inicialmente.