Universidad de las Ciencias Informáticas
Facultad 1

Título: Propuesta de Arquitectura para el Sistema de Control de Acceso a Personas y Vehículos

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autora: Lis Marrero García

Tutor: Roberlán Rodríguez Sánchez

"Año 50 de la Revolución"

Junio 6 del 2008

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

_____	_____
Lis Marrero García	**Roberlán Rodríguez Sánchez**
Autora	Tutor

A aquellos que dieron su vida, para hacer posible que jóvenes de ascendencia humilde lograran estudios superiores en forma gratuita.

A Fidel, quien hizo realidad un sueño jamás soñado. Creando esta maravillosa Universidad, donde hoy me gradúo.

A mi Patria, a nuestra Revolución, a mi querida madre, a mi familia toda, a este conjunto que fue mi sostén ideológico y social desde que comencé a aprender mis primeras letras.

RESUMEN

En el presente trabajo se realiza una propuesta de arquitectura para el Sistema de Control de Acceso de Personas y Vehículos basada en software libre. Durante el desarrollo del trabajo se analizan las principales tendencias, tecnologías y metodologías actuales, se definen los principales requerimientos del sistema y se obtiene como resultado el diseño de la arquitectura para un sistema que permita el control del acceso de personas y vehículos a la Universidad de las Ciencias Informáticas. La Arquitectura para el Sistema de Control de Acceso de Personas y Vehículos debe ser flexible y que satisfaga con los requisitos (Analistas), que pueda ser construible (diseñadores y programadores) y que cumpla con los parámetros de funcionalidad, eficiencia y confiabilidad.

Palabras Claves

Arquitectura de software: Es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos, el ambiente y los principios que orientan su diseño y evolución.

TABLA DE CONTENIDOS

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 4

1.1-Introducción 4
1.2 - Conceptos de importancia. 4
1.2.1 - Arquitectura de software. 4
1.2.2 - Sistema de Control de Acceso. 4
1.3 - Historia Arquitectura de software. 5
1.4 - Experiencias anteriores vinculadas al campo de acción. 6
1.4.1 - Ámbito Internacional. 6
1.4.2 - Ámbito Nacional. 8
1.5 - Tendencias, tecnologías y metodologías actuales 10
1.5.1 - Metodologías de desarrollo de software 10
1.5.1.1 - Extreme Programming (XP) 10
1.5.1.2 - Microsoft Solution Framework (MSF) 12
1.5.1.3 - Rational Unified Process (RUP) 14
1.5.2 - Tecnologías 17
1.5.2.1 - Java2EE 18
1.5.2.2 - Framework.NET 19
1.5.3 - Patrones arquitectónicos 19
1.5.4 - Gestores de Bases de Datos 23
1.5.4.1 - PostgreSQL 23
1.5.4.2 - MySQL 24
1.5.5 - Base de Datos Embebidas. 24
1.5.5.1 - DB4O 24
1.5.5.2 - SQLite 25
1.5.6 - Ambiente de desarrollo integrado 26
1.5.6.1 - Eclipse 27
1.5.6.2 - NetBeans 27
1.5.6.3 - Visual Studio.NET 28
1.5.7 - Control de Versiones 28
1.5.8 - Herramientas CASE (Computer Aided Software Engineering) 29
1.5.8.1 - Visual Paradigm 30
1.5.8.2 - Rational Rose 32
1.5.9 - Framework 34
1.6 - Conclusiones 35
CAPÍTULO 2: CARACTERÍSTICAS DE LA ARQUITECTURA 36

2.1 - Introducción 36
2.2 – La propuesta de solución, siguiendo las indicaciones de la Dirección de Informatización 36
2.3-Requerimientos 40
2.3.1 – Requerimientos funcionales 40

2.3.2 – Requerimientos no funcionales 41
2.4-Descripción de la Arquitectura 42
2.4.1-Vistas Arquitectónicas 43
2.4.1.1- Vista de Casos de Uso 43
2.4.1.2-Vista Lógica 55
2.4.1.3-Vista de Implementación 56
2.4.1.4-Vista de Despliegue 57
2.4.1.5 -Vista de Procesos 59
2.5.1 - Estándares de código en Java 59
2.5.2 - Clases de la capa de acceso a Datos 61
2.5.3 - Clases de la capa de Servicios de Negocio 61
2.5.4 - Recursos de la Capa de Presentación 62
2.8 - Conclusiones 63
CAPÍTULO 3: EVALUACIÓN DE LA ARQUITECTURA 64

3.1- Introducción 64
3.2- Evaluando una arquitectura de software 64
3.2.1- ¿Cuándo una arquitectura puede ser evaluada? 65
3.2.2 - ¿Quiénes estan involucrados? 65
3.2.3 - Planificación de las evaluaciones 65
3.2.4 - ¿Qué resultados produce la evaluación de una arquitectura? 66
3.2.5 - ¿Cuáles son los beneficios de realizar una evaluación arquitectónica? 66
3.2.6 - Técnicas de evaluación 66
3.2.7- ¿Cómo se puede realizar una evaluación arquitectónica? 67
3.3 - Evaluando la arquitectura propuesta 68
2.4 - Seguridad 69
3.5 - Análisis Económico 70
3.4 - Conclusiones 72
CONCLUSIONES 73

RECOMENDACIONES 74

BIBLIOGRAFÍA 75

GLOSARIO 78

ANEXOS 79

ÍNDICE DE TABLAS

Tabla 2. 1 Descripción textual del caso de uso “Autenticar usuario”.....45
Tabla 2. 2 Descripción textual del caso de uso “Controlar acceso puntual de personas de la UCI”48
Tabla 2. 3 Descripción textual del caso de uso “Controlar accesos de vehículos”.....50
Tabla 2. 4 Descripción textual del caso de uso “Configurar Sistema”. 52
Tabla 2. 5 Descripción textual del caso de uso “Actualizar Lector” 54
Tabla 2. 6 Descripción textual del caso de uso “Actualizar Base de Datos Local”.....54

ÍNDICE DE FIGURAS

Figura 1 Arquitectura en tres Capas 20
Figura 2 Modelo Vista Controlador (MVC) 22
Figura 3 db4o Replication System 39
Figura 4 Vistas de la arquitectura propuesta por RUP. 43
Figura 5 Modelo de Casos de Uso Arquitectónicamente Significativos 44
Figura 6 Modelo de Diseño. 55
Figura 7 Paquete de componentes por capa. 56
Figura 8 Diagrama de Despliegue. 58
Figura 9 Diagrama de clase del diseño "Configurar Sistema" 82
Figura 10 Diagrama de clase del diseño "Controlar Acceso puntual de Personas de la UCI" 88
Figura 11 Diagrama de clase del diseño "Controlar Acceso de Vehículo" 94
Figura 12 Diagrama de clase del diseño "Actualizar Lector" 102
Figura 13 Diagrama de clase del diseño "Actualizar Base de Datos Local 110

INTRODUCCIÓN

En la actualidad el campo de la informática a nivel mundial ha venido sufriendo cambios de manera que provocan un constante desarrollo en el ámbito tecnológico, lo que ha traído consigo que al país le sea un poco difícil mantenerse al margen de tal desarrollo.

Cuba a partir de la caída del campo socialista y el continuo bloqueo trajo como consecuencia que la economía se viera afectada, por lo que además el desarrollo tecnológico fue una de las esferas que se afectaron en gran medida, la introducción de mejoras tecnológicas se vieron frenadas y las que se usaban en aquel momento caducaron con el paso del tiempo. El país en aras de revertir esta situación ha invertido recursos de manera que la economía lo permitiera.

Durante los inicios de este siglo, el país se ha enmarcado en un proceso de informatización de la sociedad cubana, lo que ha traído con ello la necesidad de crear sistemas eficientes que respondan al desarrollo de las nuevas tecnologías, que como parte del los avances económicos que va teniendo el país, se van introduciendo.

A partir del calor de la batalla de ideas que lleva el país se han venido desarrollando diferentes programas en aras de levantar el nivel cultural y económico del país, de ahí que se han creado diferentes escuelas con características propias , la Universidad de la Ciencias Informáticas no se encuentra exenta de ello, es un proyecto donde posee una gran cantidad de estudiantes, profesores y trabajadores de todos los rincones del país y una gran suma de medios materiales que presentan gran valor en el mercado mundial, entre ellos están las computadoras, televisores, teléfonos, etc.…

Debido a tan numerosa población la Universidad de Ciencias Informáticas (UCI) recibe una gran cantidad de accesos diarios, tanto de trabajadores como visitantes, esperados y no esperados, y controlar la entrada y salida de todas estas personas es una tarea un poco difícil, lo que se desea es llevar un control estricto de la seguridad y protección física, lo cual se realiza de forma manual; para ello en la universidad se desarrolló en años anteriores un software con tecnología .Net, brindando una solución al problema existente.

La arquitectura del sistema existente no cumple con los nuevos requerimientos y la política de migración a software libre establecidas por la UCI, por lo que no se puede utilizar.

De esta situación surge el siguiente ***problema científico*:**

¿Qué Arquitectura utilizar sobre software libre para el Sistema de Control de Acceso de Personas y Vehículos para lograr módulos flexibles con los nuevos requerimientos?

Teniendo en cuenta el problema planteado se define como **objeto de estudio:**

El Sistema de Control de Acceso de Personas y Vehículos de la Universidad de las Ciencias Informáticas (UCI).

Por consiguiente, el **campo de acción** lo va a componer: La arquitectura del Sistema de Control de Acceso de Personas y Vehículos de la Universidad de Ciencias Informáticas.

Dada las condiciones se plantea como **objetivo general** de la investigación: Definir la arquitectura del Sistema de Control de Acceso de Personas y Vehículos, teniendo en cuenta su desarrollo en Software libre y los nuevos requerimientos

Con esta investigación se quiere **defender la idea** de que si se define una correcta y adecuada arquitectura del Sistema de Control de Acceso de Personas y Vehículos de la Universidad de Ciencias Informáticas entonces se obtendrá un sistema flexible, robusto y con gran calidad en el mercado de software libre.

Según el objetivo planteado para la investigación se define el siguiente conjunto **de tareas** para lograr su cumplimiento**:**

- Realización un estudio del estado del arte acerca de softwares similares al proyecto de Control de Acceso.
- Realización un estudio del actual del Sistema de Control de Acceso.
- Investigación de las herramientas en software libre que podrán ser usadas.
- Intercambio de ideas con personas especializadas en el tema de las arquitecturas.
- Definición de la nueva arquitectura ajustándose a las indicaciones propuestas por la dirección de Informatización.
- Definición del framework y del entorno de desarrollo (IDEs) más apropiado para el

desarrollo del sistema.

- Definición la seguridad del sistema.
- Definición del estándar de código a utilizar.
- Realización la ingeniería de software correspondiente.
- Estudio del análisis económico de la nueva arquitectura propuesta.

Métodos teóricos:

Analítico- sintético: para el procesamiento de la información y arribar a las conclusiones de la investigación, así como para precisar las características del modelo arquitectónico propuesto.

Análisis histórico - lógico: Para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos.

Inductivo- deductivo: A partir del estudio de distintos estilos y modelos de arquitectura arribar a proposiciones de estilos de arquitecturas específicas.

Método Sistémico: Para determinar los componentes y definir las relaciones entre estos.

Métodos empíricos:

Observación: Para la percepción selectiva de las restricciones y propiedades del sistema, y sistémica para el control de la evolución de la arquitectura inicial.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1-Introducción

En este capitulo se hace un estudio de diferentes conceptos relacionados con la problemática. Se analiza además la arquitectura del software de proyectos de control de acceso de Cuba y el mundo, para tomar experiencias al respecto. Finalmente se describen las principales tendencias, tecnologías y herramientas a tener en cuenta para diseñar una buena arquitectura de software, con el fin proponer las más adecuadas para dar solución al problema planteado.

1.2 - Conceptos de importancia.

1.2.1 - Arquitectura de software.

Una Arquitectura de Software, también denominada Arquitectura lógica, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información.

La Arquitectura de Software establece los fundamentos para que analistas, diseñadores, programadores, etc.; trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación ente ellos. Toda arquitectura debe ser implementable en una arquitectura física, que consiste simplemente en determinar qué computadora tendrá asignada cada tarea. (1)

1.2.2 - Sistema de Control de Acceso.

Un Sistema de Control de Acceso es aquel que permite flexibilidad y confiabilidad para el control y monitoreo de accesos, control de tráfico y movimientos en sus instalaciones, permitiendo la captura de datos del empleado o visitante, por identificación de huella dactilar, credencial, voz o firma, creando un expediente electrónico donde se almacenan los datos del usuario que interactúa con el sistema.

Un Sistema de Control de Acceso puede mantener interrelación con sistemas de alarmas y alertas tales como circuito cerrado de televisión (CCTV), alarmas sonoras, alarmas visuales, etc. (2)

1.3 - Historia Arquitectura de software.

Cada vez que se narra la historia de la arquitectura de software (o de la ingeniería de software, según el caso), se reconoce que en un principio, hacia 1968, Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera [Dij68a].

En la conferencia de la NATO de 1969, un año después de la sesión en que se fundara la ingeniería de software, P. I. Sharp formuló estas sorprendentes apreciaciones comentando las ideas de Dijkstra. Sharp continúa su alegación afirmando que con el tiempo probablemente llegue a hablarse de “la escuela de arquitectura de software de Dijkstra” y se lamenta que en la industria de su tiempo se preste tan poca o ninguna atención a la arquitectura.

En la misma época, otro precursor importante, David Parnas, demostró que los criterios seleccionados en la descomposición de un sistema impactan en la estructura de los programas y propuso diversos principios de diseño que debían seguirse a fin de obtener una estructura adecuada. Parnas desarrolló temas tales como módulos con ocultamiento de información [Par72], estructuras de software [Par74] y familias de programas [Par76], enfatizando siempre la búsqueda de calidad del software, medible en términos de economías en los procesos de desarrollo y mantenimiento. Aunque Dijkstra, con sus frases lapidarias y memorables, se ha convertido en la figura legendaria por excelencia de los mitos de origen de la AS, Parnas ha sido sin duda el introductor de algunas de sus nociones más esenciales y permanentes.

En la década de 1980 se perfeccionaron las técnicas descriptivas y las notaciones formales, permitiéndonos razonar mejor sobre los sistemas de software.

La década de 1990 fue sin duda la de la consolidación y diseminación de la AS en una escala sin precedentes. En la misma década, demasiado pródiga en acontecimientos, surge también la programación basada en componentes, que en su momento de mayor impacto impulsó a algunos arquitectos mayores, como Paul Clements [Cle96b], a afirmar que la AS promovía un modelo que debía ser más de integración de componentes pre-programados que de programación. Un segundo gran tema de la época fue el surgimiento de los patrones, cristalizada en dos textos fundamentales, el de la Banda de los Cuatro en 1995 [Gof95] y la serie POSA desde 1996 [BMR+96]. Como quiera que sea, la AS de este período realizó su trabajo de homogeneización de la terminología, desarrolló la tipificación de los estilos arquitectónicos y elaboró lenguajes de descripción de arquitectura (ADLs). También se consolidó la concepción de las vistas arquitectónicas, reconocidas en todos y cada uno de los frameworks generalizadores que se han propuesto (4+1, TOGAF, RM/ODP, IEEE).

En el siglo XXI, la AS aparece dominada por estrategias orientadas a líneas de productos y por establecer modalidades de análisis, diseño, verificación, refinamiento, recuperación, diseño basado en escenarios, estudios de casos y hasta justificación económica, redefiniendo todas las metodologías ligadas al ciclo de vida en términos arquitectónicos. (3)

1.4 - Experiencias anteriores vinculadas al campo de acción.

1.4.1 - Ámbito Internacional.

En las últimas décadas ha existido un gran interés a nivel mundial por parte de las diferentes instituciones de contar con sistemas informáticos que se encarguen de la identificación y el control de acceso, estos sistemas adecuándolos a los intereses propios de cada institución, los mismos en conjuntos encaminados a lograr una integridad entre ellos a partir de diferentes aplicaciones. Para el desarrollo de estos sistemas se hace necesaria una arquitectura que soporte su buen funcionamiento.

Control Biométrico de Personal

En Argentina se creó un Software llamado Control Biométrico de Personal. Este brinda tanto el monitoreo total de acceso a sectores así como el control de ausentismo del personal. Identificación por lector de huellas digitales; el sistema de la terminal chequea con la base de datos y produce el consiguiente registro. El software central permite cargar la base de empleados, incluir su correspondiente foto en la ficha, indicar horarios laborales (fijos o rotativos), exportar la base de registro de entradas a programas de liquidación de sueldos y jornales. El acceso / control de individuos se puede habilitar de manera unitaria, o bien creando grupos de personas con determinados privilegios - lo que facilita su implementación en organizaciones de gran envergadura -. Dispone de módulos opcionales que le permiten generar reportes personalizados y estadísticas. Utiliza un lector conectado a una PC.

Su arquitectura esta conformada por un diseño web clásico, con rápida carga y excelente gusto en cuanto a estilo gráfico, el cual fue diseñado en lenguajes html, flash, asp y php con conectividad a bases de datos SQL Server / MySQL. (4)

Fisterra

Fisterra es un proyecto de Software Libre que tiene como objetivo la creación de una plataforma de desarrollo de aplicaciones de gestión empresarial, y la implementación sobre esta de soluciones verticales. Fisterra se realiza una propuesta diferente, innovadora, y totalmente libre, que utiliza tecnologías GNOME y una arquitectura compleja y flexible, además constituye por si mismo un completo framework de desarrollo de aplicaciones.

Fisterra esta basada en una arquitectura de tres capas, con un servidor funcionando como middleware; la aplicación del patrón MVC de una forma estricta para el diseño de la interfaz; y la utilización de un contenedor de objetos de negocio, llamados EGBs

(Enterprise Gnome Beans), el cual se encarga de su inicialización y liberación cuando estos ya no sean requeridos.
Para el almacenamiento persistente, Fisterra utiliza la biblioteca libGDA (que aisla el gestor concreto utilizado del resto del sistema), y el gestor de bases de datos relacionales PostgreSQL. (5)

1.4.2 - Ámbito Nacional.

En Cuba el desarrollo informático se ha ido incrementado considerablemente, pues existe conciencia de la valía y significado en la elevación de la productividad que ello implica, así también de las necesidades de nuevos sistemas que cumplan los requerimientos específicos de los procesos empresariales para un mejor desempeño de las entidades.

SIGEP (Sistema de Gestión Penitenciaria)

Es un sistema informático integral cubano creado en software libre donde tiene como propósito gestionar las actividades penitenciarias, abarcar las áreas de control penal, clasificación y tratamiento, salud integral, custodia, control logístico, administración de la Dirección General y la gestión de sus unidades de apoyo así como el acompañamiento post penitenciario.

Dentro de su arquitectura utilizan como ambiente de desarrollo; plataforma Java, Tomcat como contenedor Web, Subversión como controlador de versiones, como base de dato Oracle y como herramientas de modelado al Visual Paradigm y al Erwin. El eclipse como entorno integrado de desarrollo y como framework utilizan el spring, JUnit, Aspecto, JfreeChart, entre otros.

Presenta una arquitectura multicapa donde se encuentran la capa de Acceso a Datos que contiene un lenguaje de consultas de alto nivel, una arquitectura de caché (Objetos y Consultas), herramientas de desarrollo, ficheros de mapeos flexibles e intuitivos y soporte para diferentes tipos de pruebas. La capa de Servicios de Negocio la cual es muy sencilla en el proceso de desarrollo, mantenimiento, integridad de aplicaciones e interrelación con

otros sistemas. Por ultimo esta la capa de Presentación la cual contiene Ajax que esta a su vez consiste en HTML, tecnología Javascript, DHTML y DOM; todas estas tecnologías trabajan juntas para hacer extremadamente eficiente el desarrollo Web obteniendo una transformación
en las pasadas interfaces Web en interactivas aplicaciones Ajax.
Se utiliza el patrón Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos

En los aspectos de seguridad utilizan la autenticación del usuario mediante el usuario, contraseña, dirección IP del equipo y la verificación humana. La seguridad se define y se maneja independiente de la aplicación, se utiliza también canales seguros y seguridad en los servicios.

Sistema de Control de Acceso para la Universidad de la Ciencias Informáticas (UCI)
El Sistema de Control de Acceso fue creado en la UCI en el año 2003, este sistema se encarga de garantizar la seguridad y protección de la Universidad. Este brinda un total monitoreo de los accesos, tanto de estudiantes, trabajadores y personal ajeno a la Universidad como de vehículos. Identificación mediante solapín y el código de barras impreso al dorso de este, el sistema verifica la validez de la credencial y que no ocurran salidas o entradas consecutivas de una misma credencial (acción conocida como *passback*); para el caso de las personas que pertenecen de una forma u otra a la UCI; en caso contrario se registran otros datos.
Dentro de su arquitectura utilizan como plataforma .NET, SQL Server como Gestor de Base de Datos por su fortaleza y capacidad para grandes volúmenes de información. Se utilizó SQLite para la base de datos local en las puertas, porque es muy rápido y la ventaja fundamental es que permite utilizar un amplio subconjunto del lenguaje estándar SQL.

1.5 - Tendencias, tecnologías y metodologías actuales

1.5.1 - Metodologías de desarrollo de software

En un proyecto de desarrollo de software la metodología define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo. No existe una metodología de software universal. Cada equipo de desarrollo escoge la metodología según las características de su proyecto, por lo que es importante determinar el alcance del proyecto antes de escoger la metodología que se va a usar en el desarrollo del mismo. A continuación se mencionan algunas metodologías de desarrollo que existen, dando una explicación sobre las mismas.

1.5.1.1 - Extreme Programming (XP)

XP (Extreme Programming, por sus siglas en inglés) es una metodología de desarrollo de software, de las más exitosas en la actualidad utilizadas para proyectos de corto plazo y corto equipo. La metodología consiste en una programación rápida o extrema, donde el usuario final forma parte del equipo.

Características de XP, la metodología se basa en:

- Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir. Es como si nos adelantáramos a obtener los posibles errores.

- Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

- Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.
- El cliente o el usuario se convierte en miembro del equipo.

Derechos del Cliente

- Decidir que se implementa
- Saber el estado real y el progreso del proyecto
- Añadir, cambiar o quitar requerimientos en cualquier momento
- Obtener lo máximo de cada semana de trabajo
- Obtener un sistema funcionando cada 3 o 4 meses

Derechos del Desarrollador

- Decidir como se implementan los procesos
- Crear el sistema con la mejor calidad posible
- Pedir al cliente en cualquier momento aclaraciones de los requerimientos
- Estimar el esfuerzo para implementar el sistema
- Cambiar los requerimientos en base a nuevos descubrimientos

Lo fundamental en este tipo de metodología es:

- La comunicación, entre los usuarios y los desarrolladores.

- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

1.5.1.2 - Microsoft Solution Framework (MSF)

MSF (Microsoft Solution Framework, por sus siglas en inglés) es una metodología de desarrollo de software que controla la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. Puede ser adaptada a proyectos de cualquier dimensión y usada para desarrollar soluciones sobre cualquier tecnología.
MSF tiene las siguientes características:

- Adaptable: es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- Escalable: puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
- Flexible: es utilizada en el ambiente de desarrollo de cualquier cliente.
- Tecnología Agnóstica: porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación.

- Modelo de Arquitectura del Proyecto: Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.

- Modelo de Equipo: Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.

- Modelo de Proceso: Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.

- Modelo de Gestión del Riesgo: Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.

- Modelo de Diseño del Proceso: Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.

Modelo de Aplicación: Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores

1.5.1.3 - Rational Unified Process (RUP)

RUP (Rational Unified Process, por sus siglas en inglés) es un proceso de desarrollo de software y junto con UML (Lenguaje de Modelado Unificado, por sus siglas en inglés) constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. RUP divide su ciclo de desarrollo en cuatro fases y ha agrupado sus actividades en 9 flujos de trabajo.

- **Inicio**: Esta fase tiene como objetivo determinar la visión del proyecto.

- **Elaboración**: En esta etapa el objetivo es definir la arquitectura del sistema.

- **Construcción**: En esta etapa el objetivo es llegar a obtener un producto listo para su utilización que está documentado y tiene un manual de usuario.

- **Transición**: El objetivo es llegar a obtener el *release* ya listo para su instalación. Puede implicar reparación de errores.

Flujos de Trabajo.

- **Modelamiento del negocio:** Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- **Requerimientos:** Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.

- **Análisis y diseño:** Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.

- **Implementación:** Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.

- **Prueba (Testeo):** Busca los defectos a los largo del ciclo de vida.

- **Instalación:** Produce *release* del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.

- **Administración del proyecto:** Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.

- **Administración de configuración y cambios:** Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.

- **Ambiente:** Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

Cada flujo de trabajo tiene como resultado un modelo propuesto por RUP:

- Modelo de Casos de Uso del Negocio.

- Modelo de Objetos del Negocio.

- Modelo de Casos de Uso.

- Modelo de Diseño.

- Modelo de Despliegue.

- Modelo de Datos.

- Modelo de Implementación.

- Modelo de Pruebas.

El modelo de casos de uso, el modelo de diseño, el modelo de despliegue y el modelo de implementación son los modelos más importantes para describir la arquitectura de un sistema teniendo en cuenta que son los que proporcionan el desarrollo de las vistas de arquitectura.

El ciclo de vida de RUP tiene tres características fundamentales:

Guiado por casos de uso. Los casos de uso reflejan las necesidades de los futuros usuarios, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. Los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso.

Centrado en la arquitectura. La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los casos de uso relevantes desde el punto de vista de la arquitectura (casos de uso arquitectónicamente significativo). El modelo de arquitectura se representa a través de vistas en las que se incluyen los diagramas de UML.

Iterativo e incremental. RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros.

Es práctico dividir el trabajo en partes más pequeñas o miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos

en los flujos de trabajo, y los incrementos, al crecimiento del producto. Cada iteración se realiza de forma planificada es por eso que se dice que son miniproyectos.

UML (Lenguaje Unificado de Modelado)

UML, por sus siglas en inglés, *Unified Modeling Language*: es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa (Lengua de Modelación Unificada), no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, sin embargo, la orientación a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas. (6)

1.5.2 - Tecnologías

1.5.2.1 - Java2EE

Java Platform, Enterprise Edition o Java EE (anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE es una plataforma de programación; parte de la Plataforma Java; para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

Java EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son conformes *a Java EE* no obstante sin un estándar de ISO o ECMA.

Java EE permite al desarrollador crear una Aplicación de Empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores. Otros de los beneficios de JEE es que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes.

La Plataforma Java se compone de un amplio abanico de tecnologías, cada una de las cuales ofrece una parte del complejo de desarrollo o del entorno de ejecución en tiempo real. Por ejemplo, los usuarios finales suelen interactuar con la máquina virtual de Java y el conjunto estándar de bibliotecas. Además, las aplicaciones Java pueden usarse de forma variada, como por ejemplo ser incrustadas en una página Web. Para el desarrollo de aplicaciones, se utiliza un conjunto de herramientas conocidas como JDK (Java Development Kit, o herramientas de desarrollo para Java).

Una de las principales características que favoreció el crecimiento y difusión del lenguaje Java es su capacidad de que el código funcione sobre cualquier plataforma de software y hardware. Esto significa que el mismo programa escrito para Linux puede ser ejecutado en Windows sin ningún problema. Además es un lenguaje orientado a objetos, independiente de plataforma, brinda un gran nivel de seguridad, capacidad multihilo y gran rendimiento, lo que permite resolver los problemas en la complejidad de los sistemas. (7)

1.5.2.2 - Framework.NET

El entorno de desarrollo de Microsoft .NET está formado por .NET Framework y Visual Studio .NET

.NET es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones.

El .NET Framework de Microsoft es código administrado modelo de programación para crear aplicaciones en Windows clientes, servidores y móviles o dispositivos embebidos. Aunque

.NET no es totalmente libre no se puede dejar de mencionar ya que es una tecnología muy poderosa.

El Framework de .NET es un entorno de desarrollo distribuido, en el cual se reúnen en conjunto lenguajes y servicios que simplifican el desarrollo y ejecución de aplicaciones .NET. Además de soportar múltiples lenguajes de programación y, es posible desarrollar cualquier tipo de aplicación con cualquiera de estos lenguajes: C# (C Sharp), Visual Basic, C++, J# (Java #), Jscript.

Es multiplataforma, debido a que posee el CLR (Common Language Runtime, es decir, el Motor Común de Ejecución) que es el centro neurálgico del Framework de .NET encargado de gestionar la ejecución de las aplicaciones, aplicar parámetros de seguridad y ejecutar el denominado recolector de basuras. Un programa .NET podrá ser compilado y ejecutado en cualquier plataforma que incluya un CLR. (8)

1.5.3 - Patrones arquitectónicos

Arquitectura de N Capas

“En [GS94] Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior.”

En la actualidad muchos sistemas de información están basados en arquitecturas de dos capas, denominadas generalmente nivel de aplicación y nivel de la base de datos. Este

estilo tiene algunos inconvenientes como es la recarga del nivel de aplicaciones. Es por ello que existe una fuerte y avanzada tendencia a adoptar arquitectura en tres capas. (Ver Figura.1). La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. (9)

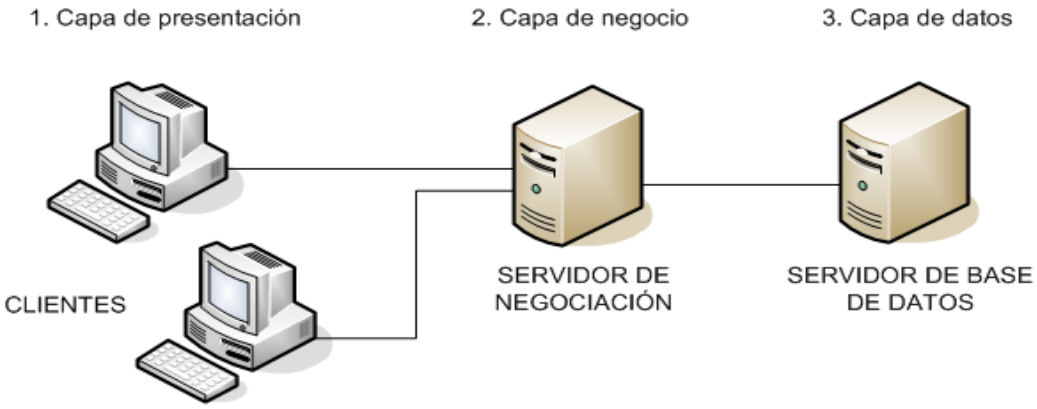


Figura 1 Arquitectura en tres Capas

Las tres capas que propone esta arquitectura son:

- **Capa de presentación**: es la que ve el usuario (hay quien la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio.

- **Capa de negocio**: es donde residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar

datos de él.

- **Capa de datos**: es donde residen los datos. Está formada por uno o más gestor de bases de datos que realiza todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Este patrón puede ser difícil a la hora de definir qué componentes ubicar en cada una de las capas, sin embargo mejora el soporte del sistema y facilita la localización de errores. [9]

Modelo-Vista-Controlador (MVC)

El MVC ha sido propio de las aplicaciones en Smalltalk por lo menos desde 1992, antes que se generalizaran las arquitecturas en capas múltiples. Un propósito común en numerosos sistemas es el de tomar datos de un almacenamiento y mostrarlos al usuario. Luego que el usuario introduce modificaciones, las mismas se reflejan en el almacenamiento. Dado que el flujo de información ocurre entre el almacenamiento y la interfaz, una tentación común, un impulso espontáneo (hoy se llamaría un anti-patrón) es unir ambas piezas para reducir la cantidad de código y optimizar la performance. Un problema es que las aplicaciones tienden a incorporar lógica de negocios que van más allá de la transmisión de datos. El patrón conocido como Modelo-Vista-Controlador (MVC), ver fig.2, separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

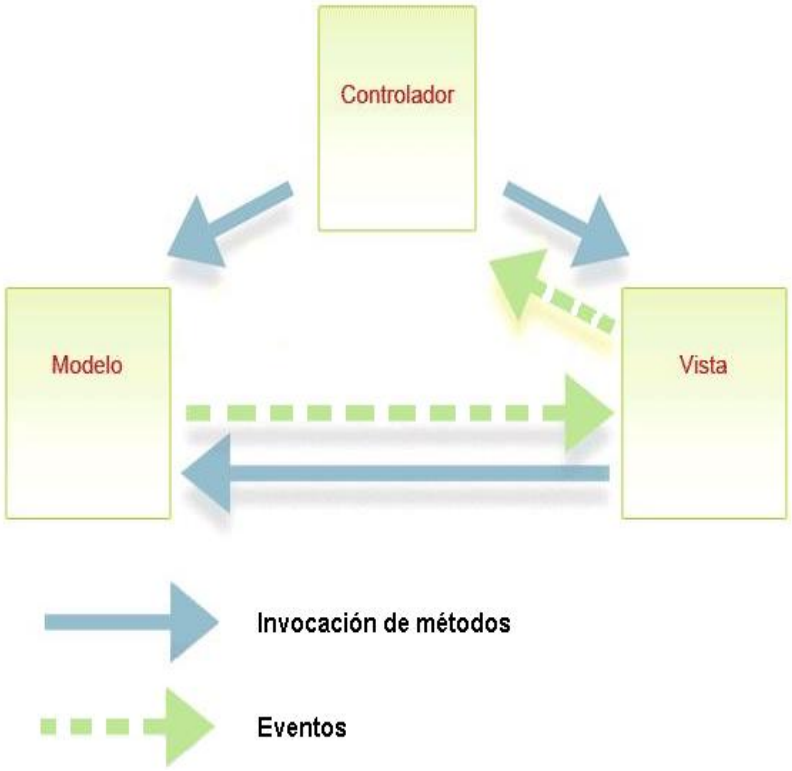


Figura 2 Modelo Vista Controlador (MVC)

- Modelo: El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

- Vista: Maneja la visualización de la información.

- Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y a la vista para que cambien según resulte apropiado. (10)

1.5.4 - Gestores de Bases de Datos

Los Sistemas de gestión de base de datos son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. (11)

1.5.4.1 - PostgreSQL

El PostgreSQL es un poderoso sistema manejador de bases de datos, es decir, un sistema diseñado para administrar grandes cantidades de datos, que tiene la fama de ser la base de datos de código abierto (Open Source) más avanzada del mundo.

PostgreSQL tiene más de 15 años de desarrollo activo y se ha ganado la reputación de ser confiable y mantener la integridad de los datos, se ejecuta en la mayoría de los Sistemas Operativos más utilizados en el mundo incluyendo, Linux, varias versiones de UNIX y por supuesto Windows.

Ofrece muchas características modernas:

- Consultas complejas.
- Claves foráneas.
- Disparadores.
- Opiniones.
- Integridad transaccional.
- Control de concurrencia multiversión.

Además, PostgreSQL puede ser ampliado por el usuario de muchas maneras, por ejemplo mediante la introducción de nuevos

- Tipos de datos.
- Funciones.
- Operadores.
- Funciones agregadas.
- Indice métodos.

- Lenguas de procedimiento.

Y debido a la licencia liberal, PostgreSQL puede ser utilizado, modificado y distribuido por todo el mundo de forma gratuita para cualquier propósito, ya sea privado, comercial o académica. (12)

1.5.4.2 - MySQL

MySQL es un sistema de gestión de bases de datos relacionales. Una base de datos relacional almacena datos en tablas separadas en lugar de poner todos los datos en un gran almacén, lo que añade velocidad y flexibilidad. La parte SQL de "MySQL" se refiere a "Structured Query Language". SQL es el lenguaje estandarizado más común para acceder a bases de datos. MySQL es Open Source, lo que quiere decir que es posible para cualquiera usar y modificar el software. Cualquiera puede bajar el software MySQL desde Internet y usarlo sin pagar nada. Si lo desea, puede estudiar el código fuente y cambiarlo para adaptarlo a sus necesidades. El servidor de base de datos MySQL es muy rápido, fiable y fácil de usar. MySQL Server trabaja en entornos cliente/servidor o incrustados. Además, funciona en diferentes plataformas y fue escrito en C y en C++. (13)

1.5.5 - Base de Datos Embebidas.

1.5.5.1 - DB4O

DB4O es un novedoso motor de base de datos orientada a objetos. Sus siglas se corresponden con la expresión "DataBase 4 (for) Objects", que a su vez es el nombre de la compañía que lo desarrolla: db4objects, Inc.

Las claves innovadoras de este producto es su alto rendimiento (sobre todo en modo embebido) y el modelo de desarrollo que proporciona a las aplicaciones para su capa de acceso a datos, el cual propugna un abandono completo del paradigma relacional de las bases de datos tradicionales.

Deja de existir un lenguaje SQL de consultas/modificaciones para pasar a crearse sistemas de consulta por métodos delegados y actualización/creación/borrado automático de entidades mediante código compilable.

Se elimina la necesidad de representar el modelo de datos de la aplicación en dos tipos de esquemas: modelo de objetos y modelo relacional. Ahora el esquema de datos del dominio viene representado por la implementación que se realice del diagrama de clases. Si se quiere desarrollar software libre con esta librería, su uso no conlleva ningún coste por licencia, db4o es “open source”. (14)

Tiene algunas características interesantes:

- No hay diferencia de impedancia: los objetos se almacenan tal y como son.
- Manejo automático del esquema de base de datos.
- No hay que cambiar las clases para poder almacenarlas.
- Uniones de datos automatizadas. (15)

1.5.5.2 - SQLite

SQLite es un sistema de gestión de bases de datos relacional compatible con ACID, y que está contenida en una relativamente pequeña (~500Kb) librería en C SQLite es un proyecto de dominio público.

A diferencia de los sistemas de gestión de base de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la librería SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

SQLite en su versión 3 usa un tipo de sistema inusual. En lugar de asignar un tipo a una columna como en la mayor parte de los sistemas de bases de datos SQL, los tipos se asignan a los valores individuales. Por ejemplo, se puede insertar un string en una columna de tipo entero (a pesar de que SQLite tratará en primera instancia de convertir la cadena en un entero). Algunos usuarios consideran esto como una innovación que hace que la base de datos se mucho más útil, sobre todo al ser utilizada desde un lenguaje de scripting de tipos dinámicos. Otros usuarios lo ven como un gran inconveniente, ya que la técnica no es portable a otras bases de datos SQL.

SQLite es una pequeña librería programada en lenguaje C que implementa un completo motor de base de datos que no precisa configuración. Es muy rápido y la ventaja fundamental es que permite utilizar el lenguaje estándar SQL.
Se destaca, además por su velocidad, por su versatilidad. Es Software Libre por lo tanto el código fuente es del dominio público y licencia GPL. (16)

1.5.6 - Ambiente de desarrollo integrado

Un ambiente de desarrollo integrado o en inglés *Integrated Development Environment* (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, poder utilizarse para varios.
Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Estos proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Visual Basic, Object Pascal, etcétera.
En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto, como es el caso de Smalltalk.
A continuación se caracterizan un conjunto de herramientas utilizadas en el ambiente de desarrollo integrado. (17)

1.5.6.1 - Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto independiente de una plataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente.

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés *plug-in*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. (18)

Eclipse dispone de las siguientes características:

- Editor de texto.
- Resaltado de sintaxis.
- Compilación en tiempo real.
- Pruebas unitarias con JUnit.
- Control de versiones con CVS.
- Asistentes (*wizards*): para creación de proyectos, clases, tests, etc.
- Refactorización

Asimismo, a través de "plugins" libremente disponibles es posible añadir:

Control de versiones con Subversion.

Integración con Hibernate.

1.5.6.2 - NetBeans

NetBeans es una aplicación de código abierto ("open source") diseñada para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas, haciendo uso de la tecnología Java.

Dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web, control de versiones, colaboración entre varias personas, creación de

aplicaciones compatibles con teléfonos móviles, resaltado de sintaxis y por si fuera poco sus funcionalidades son ampliables mediante la instalación de packs.

Con NetBeans IDE no solo es posible elaborar potentes aplicaciones para el Escritorio, también para la Web y para dispositivos portátiles, como móviles o Pocket PC, sin que cambie la forma de programar. La programación mediante NetBeans se realiza a través de componentes de software modulares, también llamados módulos. (19)

1.5.6.3 - Visual Studio.NET

Visual Studio .NET es un IDE (Entorno de Desarrollo Integrado, por sus siglas en ingles) desarrollado por Microsoft**.** La característica más notable del IDE es su soporte de los nuevos lenguajes .NET. Como C#, Visual Basic, C++, ASP, etc.
Visual Studio .NET es la herramienta de desarrollo multilenguaje más completa para construir e integrar rápidamente aplicaciones y servicios Web XML. Aumenta de un modo extraordinario la productividad de los desarrolladores y crea nuevas oportunidades de negocio. En su diseño se han integrado a fondo los estándares y protocolos de Internet, como XML y SOAP, por lo que Visual Studio .NET simplifica considerablemente el ciclo de vida del desarrollo de aplicaciones. (20)

1.5.7 - Control de Versiones

Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente. Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión por ejemplo:

Subversion

Es un software de sistemas de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Es software libre y se le conoce también como **svn** por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo. (21)

- El Subversion permite transparencia al eliminar y cambiar nombres de archivos**.**
- Copias ligeras sobre ramificaciones: **I**ndependientemente del número de ramificaciones creadas mantiene un árbol diferencial de cambios, minimizando así el espacio consumido en el depósito.
- Copias diferenciales de archivos binarios: Basado en el mismo principio de *copias ligeras*, Subversion es capaz de mantener un control diferencial sobre cualquier archivo binario del deposito así reduciendo el consumo de espacio, esto contrastado con CVS que requiere archivar copias completas de un archivo binario cada vez que éste cambia.

1.5.8 **- Herramientas CASE (Computer Aided Software Engineering)**

La ingeniería de sistemas asistida por ordenador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo, su objetivo es acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o mas fases del ciclo de vida del desarrollo de sistemas.

La tecnología CASE supone la automatización del desarrollo del software, contribuyendo a mejorar la calidad y la productividad en el desarrollo de sistemas de información y se plantean los siguientes objetivos:

- Permitir la aplicación práctica de metodologías estructuradas, las cuales al ser realizadas con una herramienta se consigue agilizar el trabajo.

- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento de los programas.
- Mejorar y estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes software.
- Permitir un desarrollo y un refinamiento visual de las aplicaciones, mediante la utilización de gráficos.

Automatizar:

El desarrollo del software.
La documentación.
La generación del código.
El chequeo de errores.
La gestión del proyecto.

Permitir:

La reutilización del software.
La portabilidad del software.
La estandarización de la documentación.

1.5.8.1 - Visual Paradigm

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Es un software no propietario por lo se convierte en un herramienta fundamental a tener en cuenta a la hora de ser utilizada. (22)

Características:

- Licencia Gratuita y Comercial.
- Producto de calidad.
- Soporta aplicaciones Web.
- Varios idiomas.
- Generación de código para Java y exportación como HTML.
- Fácil de instalar y actualizar.
- Compatibilidad entre ediciones.

El Visual Paradigm ofrece:

- Un entorno de creación de diagramas para UML 2.0.
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.

Posibilita la representación gráfica de los diagramas permitiendo ver el sistema desde diferentes perspectivas, como el de componentes, despliegue, secuencia, casos de uso, clase, actividad, estado, entre otros. Además, identifica requisitos y comunica información, se centra en cómo los componentes del sistema interactúan entre ellos, sin entrar en detalles excesivos, además, permite ver las relaciones entre los componentes del diseño y mejora la comunicación entre los miembros del equipo usando un lenguaje gráfico.

Tiene disponible distintas versiones: Enterprise, Professional, Standard, Modeler, Personal y Community. Facilita licencias especiales para fines académicos.

Se integra además con las siguientes herramientas Java:

- Eclipse/IBM WebSphere.
- JBuilder.
- NetBeans IDE.
- Oracle JDeveloper.
- BEA Weblogic.

Visual Paradigm es una poderosa herramienta CASE que al igual que el Rational Rose utiliza UML para el modelado, es la herramienta por excelencia para ser utilizada en un ambiente de software libre. Permite crear tipos diferentes de diagramas en un ambiente totalmente visual. Es muy sencillo de usar, fácil de instalar y actualizar. Genera código para varios lenguajes. Tiene integrado el MS Visio y es compatible con otras ediciones. Un entorno de creación de diagramas para UML 2.0.

VP es una herramienta de diseño naciente que brinda tantas posibilidades como el Rational Rose, anterior herramienta utilizada, con la diferencia que tiene compatibilidad con Linux, por lo cual se hace la herramienta primordial para obtener el objetivo. Luego del estudio llevado a cabo sobre las herramientas Case que usan UML como lenguaje de modelaje se llego a la conclusión que la herramienta más optima es el Visual Paradigm. (23)

1.5.8.2 - Rational Rose

Rational Rose es la herramienta CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) desarrollada por los creadores de UML (Booch, Rumbaugh y Jacobson), que cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables. El navegador UML de Rational Rose nos permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable. Facilita el desarrollo de un proceso cooperativo en el que todos los agentes

tienen sus propias vistas de información (vista de casos de uso, vista lógica, vista de componentes y vista de despliegue), pero utilizan un lenguaje común para comprender y comunicar la estructura y la funcionalidad (24)

Características Rational Rose:

- Mantiene la consistencia de los modelos del sistema software.
- Generación Documentación automáticamente.
- Generación de Código a partir de los Modelos.
- Ingeniería Inversa (crear modelo a partir código).
- Característica de control por separado de componentes modelo que permite una administración más granular y el uso de modelos.
- Capacidad de análisis de calidad de código.
- Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos
- Capacidad de crear definiciones de tipo de documento XML para el uso en la aplicación
- Integración con otras herramientas de desarrollo de Rational.
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo. (25)

1.5.9 - Framework

En el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Un framework representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio. (26)

Hibernate es un potente framework de mapeo objeto/relacional y servicio de consultas para Java. Es la solución ORM (Object-Relational Mapping) más popular en el mundo Java. Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada se puede generar BBDD en cualquiera de los entornos soportados: Oracle, DB2, MySQL, etcétera.
Sus principales características son:

- Permite expresar consultas utilizando SQL nativo o consultas basadas en criterios.

- Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones J2EE y contenedores web, y por supuesto también puede utilizarse en aplicaciones de escritorio.

- Puede operar proporcionando persistencia de una manera transparente para el desarrollador.

- Soporta el paradigma de orientación a objetos de una manera natural: herencia, polimorfismo, composición y el framework de colecciones de Java.

- Soporte para modelos de objetos con una granularidad muy fina Permite una gran variedad de mapeos para colecciones y objetos dependientes.

- Proporciona el lenguaje HQL en cual provee una independencia del SQL de cada base de datos, tanto para el almacenamiento de objetos como para su recuperación.

- Presenta un potente mecanismo de transacciones de aplicación llegando incluso a permitir la interacciones largas (aquellas que requieren la interacción con el usuario durante su ejecución).

- Soporta los diversos tipos de generación de identificadores que proporcionan los sistemas gestores de bases de datos así como generación independiente de la base de datos, incluyendo identificadores asignados por la aplicación o claves compuestas. (27)

1.6 - Conclusiones

En el presente capítulo se realiza un estudio de los sistemas existentes tanto a nivel internacional como nacional, los cuales presentan arquitecturas similares enfocadas al campo de acción. Este estudio aportó conocimientos para la toma de decisiones sobre la arquitectura a escoger. Además se analizan las diferentes metodologías, tendencias y tecnologías que podrían dar solución al problema planteado.

CAPÍTULO 2: CARACTERÍSTICAS DE LA ARQUITECTURA

2.1 - Introducción

La arquitectura es el esqueleto o base de una aplicación, en esta se analiza la aplicación desde varios puntos de vista. En la arquitectura aparecen los artefactos más importantes para establecer un esquema de cómo deben ser los próximos artefactos a construir, es un semi-molde al que se deben ajustar sin muchos cambios, los restantes artefactos a construir en la aplicación o solución. De obtenerse un artefacto demasiado diferente a los demás, este formaría parte de la arquitectura. En este capítulo se especifican los requerimientos funcionales y no funcionales, casos de uso, clases y diagramas que son arquitectónicamente significativos para el sistema.

2.2 – La propuesta de solución, siguiendo las indicaciones de la Dirección de Informatización

La Universidad de las Ciencias Informáticas se encuentra inmersa en una estrategia de migración a software libre de todos los sistemas que se desarrollen.
Para implementar la propuesta de arquitectura del sistema de Control de Acceso de Personas y Vehículos, basándose en los elementos antes expuestos de algunas de las herramientas y tecnologías disponibles actualmente, se decidió utilizar RUP, porque además de ser la metodología que se estudia en la Universidad de las Ciencias Informáticas presenta una abundante información; del mismo modo las fases de desarrollo que propone, el cumplimiento de hitos en cada una de las fases y el trabajo por roles garantiza que se desarrolle el proyecto de manera efectiva y eficiente. Esta metodología se usa principalmente cuando se requiere de la implementación de sistemas de larga duración.

Se trabajará con la tecnología J2EE porque proporciona actualmente una plataforma estándar para aplicaciones distribuidas. Crea un estándar en el cuál los componentes de la aplicación pueden ser distribuidos y reutilizados, adicionalmente a través de las

especificaciones de Java Connector Architecture, permite una integración de sistemas aprovechando los recursos al máximo como reutilización de código, simplificación de los procesos de desarrollo, alta escalabilidad de la aplicación e integración de aplicaciones, un mantenimiento más rápido debido a que son pequeñas unidades de código. Java se fundamentó en un estricto modelo de seguridad.

Las aplicaciones Java pueden correr en una amplia gama de sistemas operativos (desde sistemas empresariales como Windows 2000, OS/390, Solaris, HP-UX, IRIX u otras versiones de Unix hasta en sistemas orientados más a ordenadores personales como Mac OS, Windows 9x ó Linux, y en sistemas operativos para dispositivos móviles) y de arquitecturas hardware.

Como lenguaje asociado a esta plataforma la tecnología Java, se caracteriza por ser orientado a objetos, esto permite crear programas modulares y reutilizables de código; pone mucha atención a los principios de la comprobación de posibles errores, como los compiladores que son capaces de detectar muchos de los problemas que aparecen durante el primer tiempo de ejecución en otros idiomas; es seguro ya que considera la seguridad como parte de su diseño; posee la capacidad para realizar varias tareas simultáneamente dentro de un programa; y además es independiente de la plataforma, que es la capacidad para pasar fácilmente de un sistema a otro.

Se utilizará la arquitectura en tres capas que define el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Este patrón arquitectónico permite que una capa no dependa de la otra, esto facilita que cuando ocurran cambios en una de ellas las demás no se afecten y cada capa presente sus propias funcionalidades y responda por ellas.

Se utilizará como ambiente de desarrollo integrado (IDE) el NetBeans, ya que es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de

programación. Existe además un número importante de módulos para extender el IDE NetBeans 6.0, es un producto libre y gratuito sin restricciones de uso.
Con él no sólo es posible elaborar potentes aplicaciones para Escritorio, también para la Web y para dispositivos portátiles, sin que cambie la forma de programar. Se ejecuta en muchas plataformas, incluyendo Windows, Linux, Solaris y MacOS.

La herramienta de modelado propuesta será Visual Paradigm, ya que tiene dentro de sus características que es multiplataforma, portable y posee gran facilidad de uso, es la herramienta por excelencia para ser utilizada en un ambiente de software libre. Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Además permite generar código desde diagramas y generar documentación.

Como el sistema trabajará con una abundante información, se utilizará como gestor de base de datos PostgreSQL 8.2 que es un sistema diseñado para administrar grandes cantidades de datos, además representa un sistema de administración de bases de datos relacionales de código abierto que utiliza un subconjunto de instrucciones del lenguaje SQL, es multiplataforma, extensible, estable y confiable.
Como base de datos embebida está Db4o, esta realiza la función de base de datos local que almacena una copia de la base de datos principal, ambas bases de datos se sincronizarán cada cierto tiempo mediante el db4o Replication System (dRS), esto permitirá que en todo momento se tenga el control estricto de los acceso. Dicha base de datos estará instalada en los puntos de entradas, su propósito fundamental consiste en tener donde almacenar y buscar información una vez que falle la conexión entre la aplicación y el servidor de base de datos principal.
El (dRS), con el soporte de Hibernate, permite a los usuarios construir aplicaciones que sincronizan los objetos bi-direccionalmente distribuidos entre los casos de db4o líder de código abierto y objeto de base de datos común de todas las bases de datos relacionales como Postgre o MySQL. El db4o no se conecta con SQL, trabaja con objetos puros, quien se encarga de dicha conversión es Hibernate 3.2, el cual es un potente framework de mapeo objeto/relacional y servicio de consultas para Java, este permite programar

orientado a objetos pero almacenar éstos en bases de datos relacionales, es decir, los objetos que se programen, Hibernate los mapea con las tablas del mundo relacional. (28)

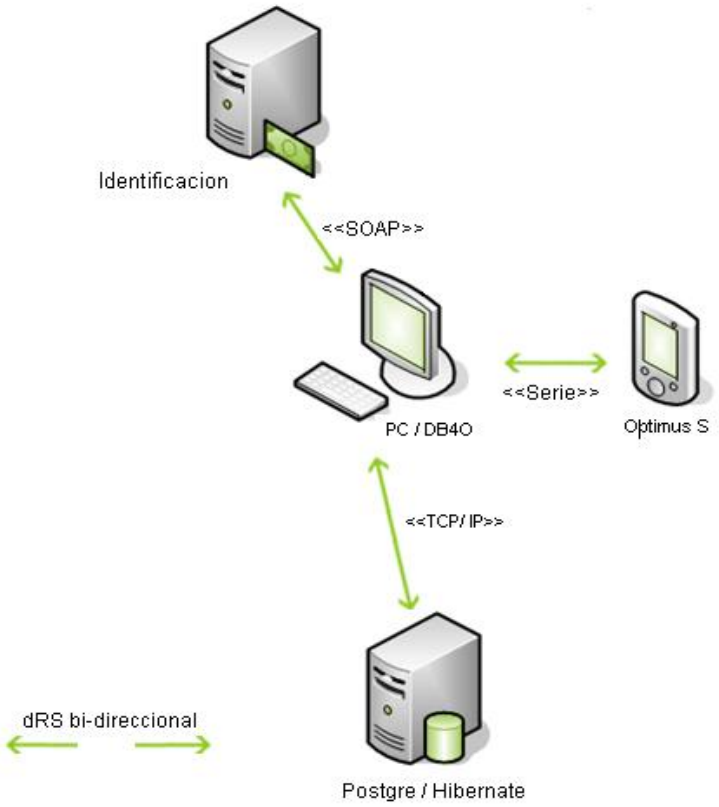


Figura 3 db4o Replication System

Como Control de versiones se tomo en cuenta Subversion, el cual sigue la historia de los archivos y directorios a través de copias y renombrados, la creación de ramas y etiquetas es una operación más eficiente; tiene costo de complejidad constante (O(1)) y no lineal (O(n)) como en CVS. Se envían sólo las diferencias en ambas direcciones (en CVS siempre se envían al servidor archivos completos), Maneja eficientemente archivos binarios (a diferencia de CVS que los trata internamente como si fueran de texto).

2.3-Requerimientos

2.3.1 – Requerimientos funcionales

RF-1 Autenticar usuario.

RF-1.1 Verificar Usuario y contraseña.

RF-1.2 Asignar Privilegios.

RF-2 Gestionar acceso puntual de personas de la UCI.

RF-2.1 Verificar la validez de las credenciales de personas de la UCI.

RF-2.2 Registrar las entradas de personas de la UCI.

RF-2.3 Registrar las salidas de personas de la UCI.

RF-2.4 Eliminar las entradas de personas.

RF-2.5 Eliminar las salidas de personas.

RF-3 Gestionar accesos de vehículos.

RF-3.1 Verificar la validez de los vehículos de forma puntual.

RF-3.2 Registrar las entradas de los vehículos.

RF-3.3 Registrar las salidas de los vehículos.

RF-3.4 Eliminar las entradas de vehículos.

RF-3.5 Eliminar las salidas de vehículos.

RF-4 Verificar PassBack.

RF- 4.1 Registrar motivo de PassBack.

RF-5 Registrar Medios.

RF-6 Mostrar Avisos

RF-7. Configurar el sistema.

RF-7.1 Configurar tiempos de sincronización de las BD.

RF-7.2 Visualizar datos de configuración del sistema.

RF-8. Actualizar Lector.

RF-9. Actualizar Base de Datos Local.

2.3.2 – Requerimientos no funcionales

- **Requerimientos de apariencia o interfaz externa:** El sistema debe tener un ambiente amigable y entendible para los usuarios finales, de forma tal que no les sea muy complicado utilizar el software.

- **Requerimientos de usabilidad:** El sistema podrá ser usado por personas autorizadas, la navegabilidad debe no debe ser muy compleja, todas las funcionalidades deben ser rápidamente accesibles por el usuario.

- **Requerimientos de rendimiento:** El tiempo de respuesta de una petición al servidor deber ser rápido ya que el sistema operara con grandes cantidades de datos.
- **Requerimientos de soporte:** Se le debe dar mantenimiento periódicamente a los servidores de bases de datos controlando la integridad de la información.

- **Requerimientos de portabilidad:** Debe tener facilidad para adaptarlo a diferentes ambientes. Independencia de la plataforma.

- **Requerimientos de seguridad y privacidad.**

    - Identificar al usuario antes de que pueda realizar cualquier acción sobre la configuración del sistema.
    - Garantizar que la información sea vista solo por personas autorizadas.
    - Verificación sobre acciones irreversibles (eliminaciones).
    - Garantía de que el sistema funcione correctamente aun cuando no haya conectividad.

- **Requerimientos de confiabilidad:** Se debe chequear la integridad de los datos.

- **Requerimientos Legales:**

  Debe cumplir con:

  - ✓ Decreto Ley – 186 / 98

  2da Sección, Artículos 5, 8, 11, Inciso D.

  - ✓ Reglamento del Decreto Ley 186 (Resolución No. 2 / 2001 del Ministerio del Interior).

  Artículo 1. Capítulo IV (Artículos 3, 4, 90)

  - ✓ El Plan de Seguridad y Protección del Centro.

- **Requerimientos de Ayuda y documentación en línea:** Se debe brindar un sistema de ayuda que explique las diferentes funcionalidades con que cuenta el sistema, además los manuales de usuario.

- **Requerimientos en el diseño y la implementación:** Para el análisis y el diseño del sistema debe ser utilizada la metodología RUP, usando el lenguaje de modelado UML y como herramienta para llevarlo a cabo el Visual Paradigm.

- **Requerimientos de software:** Se debe disponer en el servidor con Windows XP, Windows 2000 Server o 2000 Advanced Server. Se utilizará como lenguaje de programación: Java y como gestor de Base de Datos: PostgreSQL.

- **Requerimientos de hardware:** Requiere estar instalada en una PC Pentium, 256 Mb de RAM (como mínimo), Micro 2.0 o superior, 280 Mb de disco duro, una tarjeta de red de 100Mbps y el procesador de 133 MHz o superior.

2.4-Descripción de la Arquitectura

2.4.1-Vistas Arquitectónicas

En RUP, la arquitectura se compone de 4+1 vistas fundamentalmente: Vista de Casos de Uso, Vista Lógica, Vista de Procesos, Vista de Implementación y Vista de Despliegue. Estas vistas son la parte esencial de lo que se obtuvo en los flujos de Requerimientos, Análisis y Diseño e Implementación, que son las etapas que más tributan a la arquitectura. La arquitectura se representa a través de 4+1 vista (Ver figura. 4), dicho de modo que se entienda que 4 de estas vistas están regidas por una rectora: la Vista de Casos de Uso. Esto responde, además, a una de las características de RUP: Guiado por Casos de Usos.

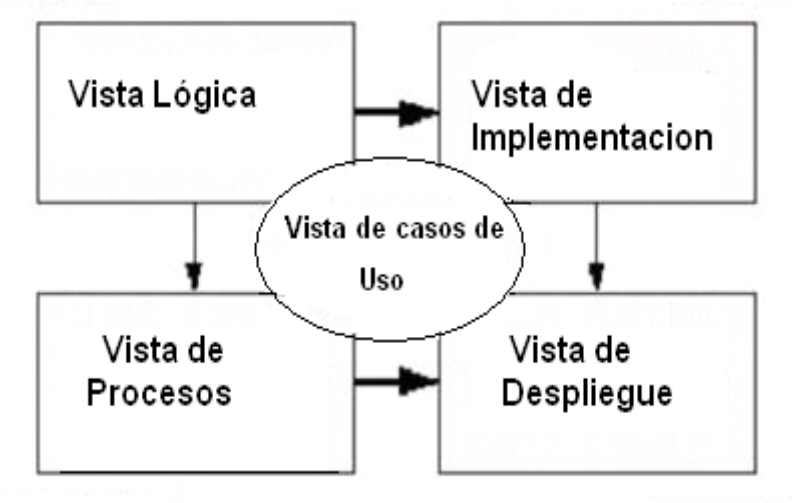


Figura 4 Vistas de la arquitectura propuesta por RUP.

2.4.1.1- Vista de Casos de Uso

La vista de casos de uso representa un subconjunto del artefacto Modelo de Casos de Uso y lista los casos de uso o escenarios del modelo de casos de uso más significativos, con las funcionalidades centrales del sistema. En esta vista se representa el diagrama de casos de uso con los casos de uso arquitectónicamente significativos para el sistema y una breve descripción de cada uno.

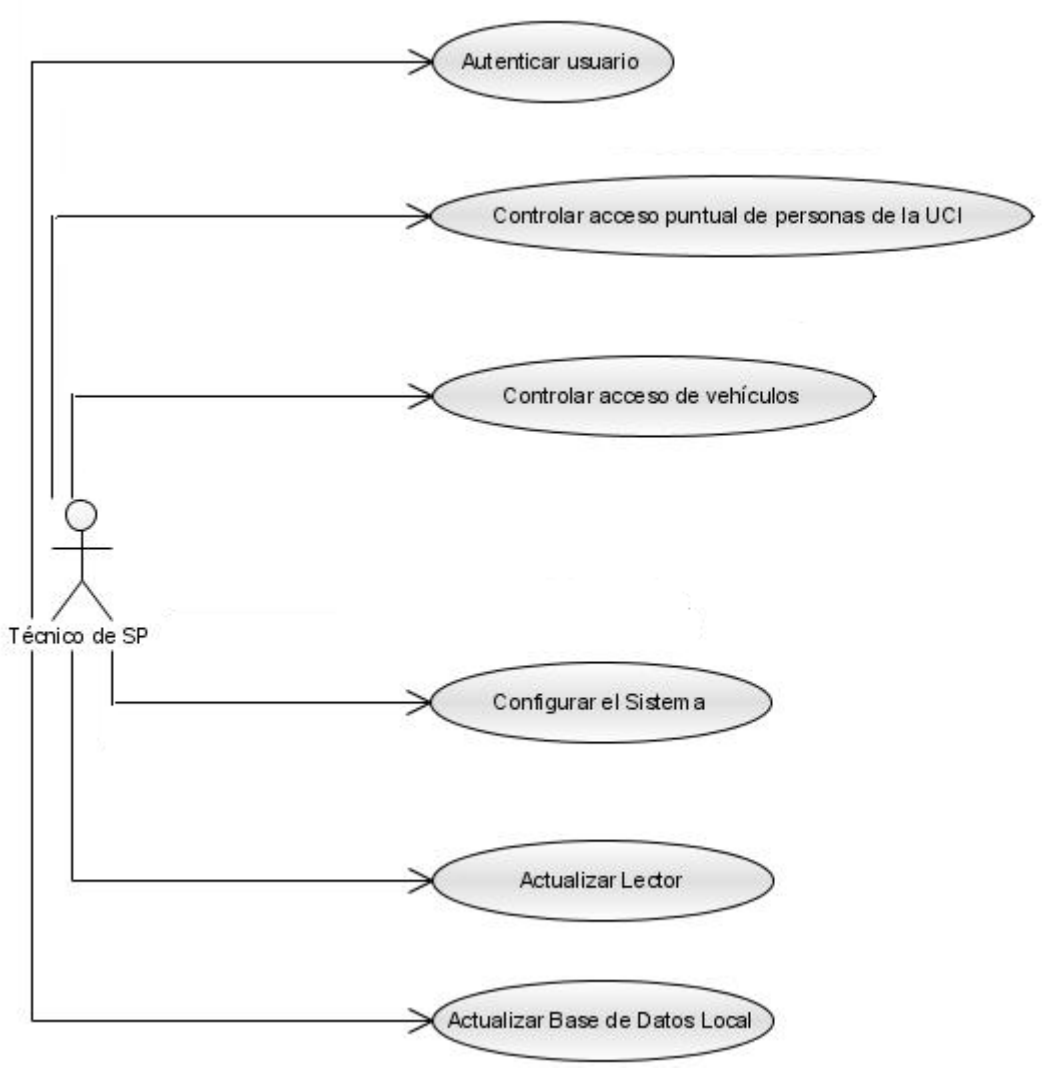


Figura 5 Modelo de Casos de Uso Arquitectónicamente Significativos

Descripción de los casos de usos del sistema arquitectónicamente más significativos.

Caso de Uso:	**Autenticar Usuario**.
Actores:	Técnico de Seguridad y protección (inicia)
Resumen:	El caso de uso se inicia cuando el Técnico SP introduce los datos que se le piden para acceder a la aplicación (usuario y contraseña), estos se verifican, finalizando así el caso de uso.
Precondiciones:	
Referencias	**RF 1, RF 1.1**

Prioridad	Alta
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El usuario introduce sus datos y elige la opción “Aceptar”. En caso contrario ver **FA1.**	2- Comprueba que el usuario y la contraseña sean correctos. En caso contrario, ver **FA2**. 3. Permite el acceso al sistema. Terminando así el caso de uso.
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
FA1	
1- Selecciona la opción “Cancelar”.	2- Se cierra la aplicación.
FA2	
	1- Muestra mensaje de error, terminando así el caso de uso.
Prototipo de Interfaz	
Poscondiciones	Se autenticó el usuario.

Tabla 2. 1 Descripción textual del caso de uso “Autenticar usuario”

Caso de Uso:	**Controlar acceso puntual de personas de la UCI**
Actores:	Técnico de Seguridad y protección (inicia)
Resumen:	El caso de uso se inicia cuando el Técnico de SP solicita controlar el acceso puntual de una persona perteneciente a la UCI, verifica la validez de los datos y registra la entrada o salida.
Precondiciones:	El Técnico de SP debe estar autenticado en el sistema.
Referencias	**RF 2, RF 2.1, RF 2.2, RF 2.3, RF 2.4, RF 2.5**
Prioridad	Alta
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1- El caso de uso se inicia cuando el Técnico SP selecciona en la ventana principal la opción “Persona” para controlar el acceso puntual. 3- Selecciona la opción deseada.	2- Muestra la interfaz con la opción “Auto Registrar“, que contiene las opciones de “Entrada”, “Salida” (desactivadas) y “Deshacer”. Y la opción “Buscar” desactivada que contiene las opciones “Entrada”, “Salida”, desactivadas y “Deshacer” y los campos: • Solapín. • CI. Y los datos vacíos: • Solapín. • Nombre. • Tipo. • CI. • Estado. • Fecha de vencimiento (Eventual). • Foto. 4- Ejecuta una de las siguientes acciones: a) Autoregistrar. Ver Sección “Autoregistrar”. b) Buscar. Ver Sección “Buscar”. c) Deshacer. Ver Sección “Deshacer”.
Secciones del Flujo Básico	
Acción del Actor	**Respuesta del Sistema**
Sección “Autoregistrar**”**	
2- Selecciona el tipo de acceso. 3- Introduce el código de barras.	1- Activa las opciones “Entrada” y “Salida” 4- Captura el código de barras. 5 Comprueba que el dato capturado (código de barras) pertenece a una persona de la UCI. En caso contrario ver FA 1. 6- Verifica PassBack. Ver CU “Verificar PassBack”. 7- Actualiza el estado de la persona. 8- Muestra los datos. • Solapín. • Nombre. • Tipo. • CI. • Estado. • Fecha de vencimiento (Eventual). • Foto. 9- Guarda la última fecha en que se

	realizó el acceso y el identificador.
Sección "Buscar"	
1- Introduce el dato por el cual desea realizar la búsqueda (solapín o CI). 2- Selecciona la opción "Buscar". 6- Selecciona el tipo de acceso.	3- Comprueba que el dato introducido pertenece a una persona de la UCI. En caso contario ver FA 1. 4- Muestra los datos. • Solapín. • Nombre. • Tipo. • CI. • Estado. • Fecha de vencimiento (Eventual). • Foto. 5- Activa las opciones "Entrada" y "Salida". 7- Verifica PassBack. Ver CU "Verificar PassBack". 8- Actualiza el estado de la persona. 9- Actualiza el campo estado de la interfaz. 10- Guarda la última fecha en que se realizó el acceso y el identificador.
Sección "Deshacer"	
2- Selecciona la opción "Aceptar". En caso contrario ver FA3.	1- Muestra un mensaje de verificación, indicando si desea cambiar el último acceso con las opciones "Aceptar" y "Cancelar". En caso contrario ver FA2. 3- Elimina el último acceso registrado. 4- Va al paso 4 del flujo normal de eventos.
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
FA1	
	1-Muestra un mensaje de error indicando que la persona no se encontró, finalizando así el caso de uso.
FA2	
	1- Muestra un mensaje indicando que no hay ningún acceso registrado.
FA3	
1- Selecciona la opción "Cancelar".	3- Va al paso 4 del flujo normal de

	eventos.
Prototipo de Interfaz	
Poscondiciones	Se registró el acceso de personas de la UCI.

Tabla 2. 2 Descripción textual del caso de uso "Controlar acceso puntual de personas de la UCI"

Caso de Uso:	**Controlar accesos de vehículos.**
Actores:	Técnico de Seguridad y protección (inicia)
Resumen:	El Caso de Uso se inicia cuando el Técnico de SP solicita controlar el acceso de vehículos, verifica la validez de los datos y registra la entrada o salida.
Precondiciones:	El Técnico de SP debe estar autenticado en el sistema.
Referencias	**RF 3, RF 3.1, RF 3.2, RF 3.3, RF 3.4, RF 3.5**
Prioridad	Alta.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El caso de uso inicia cuando el Técnico SP selecciona en la ventana principal la opción "Vehículo" para controlar el acceso.	
	2- Muestra la interfaz que contiene un campo: chapa; la opción "Buscar", y las opciones "Entrada", "Salida" (desactivadas) y "Deshacer". Y los datos vacíos: • Chapa. • Color. • Marca. • Foto del vehículo. • Foto del chofer. • Nombre del chofer.
3- Selecciona la opción deseada.	
	4- Ejecuta una de las siguientes acciones:

	a) Buscar. Ver Sección “Buscar”. b) Deshacer. Ver Sección “Deshacer”.
Secciones del Flujo Básico	
Sección “Buscar”	
Acción del Actor	**Respuesta del Sistema**
1- Introduce el dato por el cual desea realizar la búsqueda (chapa). 2- Selecciona la opción “Buscar”. 7- Selecciona la opción “Entrada” o “Salida”.	3- Comprueba que la chapa pertenece a un vehículo. En caso contrario ver FA1 4- Actualiza el estado del vehículo. 5- Muestra los datos. • Chapa. • Color. • Marca. • Foto del vehículo. • Foto del chofer. • Nombre del chofer. 6- Activa las opciones “Entrada y Salida”. 8- Verifica PassBack. Ir al CU “Verificar PassBack”. 9- Guarda la fecha del acceso, la hora y la puerta por la que ocurrió el acceso.
Sección “Deshacer”	
Acción del Actor	**Respuesta del Sistema**
2- Selecciona la opción “Aceptar”. En caso	1- Muestra un mensaje de verificación, indicando si desea cambiar el último acceso con las opciones “Aceptar” y “Cancelar”.

contrario ver FA3.	En caso contrario ver FA2. 3- Va al paso 4 del flujo normal de eventos
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
FA1	
	1-Muestra un mensaje indicando que la chapa no pertenece a ningún vehículo de la UCI.
FA2	
	1- Muestra un mensaje indicando que no hay ningún acceso registrado.
FA3	
1- Selecciona la opción “Cancelar”.	2- No elimina el último acceso registrado. 3- Va al paso 4 del flujo normal de eventos.
Prototipo de Interfaz	
Poscondiciones	Se registró el acceso de personas de la UCI.

Tabla 2. 3 Descripción textual del caso de uso “Controlar accesos de vehículos”.

.

Caso de Uso:	**Configurar el Sistema.**
Actores:	Técnico de Seguridad y protección (inicia)
Resumen:	El caso de uso inicia cuando el Técnico de SP accede a la configuración y se muestra una ventana para actualizar o cambiar los parámetros de configuración como tiempo para actualizar las BDs, nombre de la puerta, etc. Finalizando así el caso de uso.
Precondiciones:	El Técnico de SP debe estar autenticado en el sistema.
Referencias	**RF 8, RF 8.1, RF 8.2**
Prioridad	Alta
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El caso de uso inicia cuando el Técnico SP	2- Muestra una interfaz con las opciones:

selecciona en la ventana principal la opción “Configurar Sistema”. 3- Selecciona la opción deseada.	• Sincronización de Accesos de Personas. • Sincronización de Accesos de Vehículos. • Sincronización de Accesos de Visitantes. • Sincronización del listado local de Personas. • Sincronización del listado local de Vehículos. • Sincronización del listado local de Visitantes. • Sincronización automática. Con los campos para introducir los tiempos de las sincronizaciones. Y muestra la última fecha en que se hizo la sincronización, Nombre de puerta. Y las opciones “Aceptar” y “Cancelar”. 4-Ejecuta una de las siguientes acciones: a) Sincronización de Accesos de Personas. Ver Sección “Sincronización de Accesos de Personas”. b) Sincronización de Accesos de Vehículos. Ver Sección “Sincronización de Accesos de Vehículos”. c) Sincronización de Accesos de Visitantes. Ver Sección “Sincronización de Accesos de Visitantes”. d) Sincronización del listado local de Personas. Ver Sección “Sincronización del listado local de Personas”. e) Sincronización del listado local de Vehículos. Ver Sección “Sincronización del listado local de Vehículos”. f) Sincronización del listado local de Visitantes. Ver Sección “Sincronización del listado local de Visitantes”. g) Sincronización automática. Ver Sección “Sincronización Automática”.
Sección “Sincronización de Accesos de Personas”	
Acción del Actor	**Respuesta del Sistema**
2- Selecciona la opción “Aceptar”. En caso contrario ver FA1.	1- Sube los accesos de las persona de la BD local a la BD remota.
Sección “Sincronización de Accesos de Vehículos”	

Acción del Actor	**Respuesta del Sistema**
2- Selecciona la opción “Aceptar”. En caso contrario ver FA1.	1- Sube los accesos de los vehículos de la BD local a la BD remota.
Sección “Sincronización de Accesos de Visitantes”	
2- Selecciona la opción “Aceptar”. En caso contrario ver FA1.	1- Sube los accesos de los visitantes de la BD local a la BD remota.
Sección “Sincronización del listado local de Personas”	
Acción del Actor	**Respuesta del Sistema**
2- Selecciona la opción “Aceptar”. En caso contrario ver FA1.	1-Baja el listado de personas de la BD remota a la BD local.
Sección “Sincronización del listado local de Vehículos”	
Acción del Actor	**Respuesta del Sistema**
2- Selecciona la opción “Aceptar”. En caso contrario ver FA1.	1- Baja el listado de vehículos de la BD remota a la BD local.
Sección “Sincronización del listado local de Visitantes”	
2- Selecciona la opción “Aceptar”. En caso contrario ver FA1.	1- Baja el listado de visitantes de la BD remota a la BD local.
Sección “Sincronización automática”	
1- Introduce el tiempo en el que desea realizar las actualizaciones automáticas. 2- Selecciona la opción “Aceptar”. En caso contrario ver FA1.	3- Guarda en la BD local los tiempos de sincronización introducidos. Finalizando así el caso de uso.
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
FA1	
1- Selecciona la opción “Cancelar”.	2- Va a la ventana principal.
Prototipo de Interfaz	
Poscondiciones	Queda configurado el sistema.

Tabla 2. 4 Descripción textual del caso de uso “Configurar Sistema”.

<table>
<tr><td>Caso de Uso:</td><td colspan="2">Actualizar Lector</td></tr>
<tr><td>Actores:</td><td colspan="2">Técnico de Seguridad y protección (inicia)</td></tr>
<tr><td>Resumen:</td><td colspan="2">El caso de uso inicia cuando el Técnico SP solicita actualizar el lector, donde son introducidos los códigos de barra actualizados. Finalizando así el caso de uso.</td></tr>
<tr><td>Precondiciones:</td><td colspan="2">El Scanner debe ser conectado a un terminal de PC que contenga la BD local de personas, vehículos, y que contenga la aplicación. Debe existir una aplicación encargada de subir información al lector portátil.</td></tr>
<tr><td>Referencias</td><td colspan="2">RF 9</td></tr>
<tr><td>Prioridad</td><td colspan="2">Alta.</td></tr>
<tr><td colspan="3">Flujo Normal de Eventos</td></tr>
<tr><td>Acción del Actor</td><td colspan="2">Respuesta del Sistema</td></tr>
<tr><td>1- El caso de uso inicia cuando el Técnico SP selecciona en la ventana principal la opción “Actualizar Lector”.

3- Selecciona la opción “Actualizar Optimus S”. En caso contrario ver FA1.

5- Selecciona las opciones con los valores correctos para actualizar en lector.
6- Selecciona la opción “Aceptar”. En caso contrario ver FA1.</td><td colspan="2">2- Muestra la interfaz que contiene las opciones “Actualizar Optimus S” y “Cancelar”. Y dos listados (personas o vehículos) para seleccionar el dato por el cual será actualizado el lector.

4- Muestra una interfaz que contiene:
Una opción para pasar al Optimus los códigos de barra. (Permite buscar el fichero que contiene los códigos de barra)
Tipo de plataforma (escoger)
Puerto COM (escoger)
Velocidad de Trasmisión (escoger)
Y las opciones “Aceptar” y “Cancelar”.

7- Actualiza el lector. Finalizando así el caso de uso.</td></tr>
<tr><td colspan="3">Prototipo de Interfaz</td></tr>
<tr><td colspan="3">Flujos Alternos</td></tr>
<tr><td>Acción del Actor</td><td colspan="2">Respuesta del Sistema</td></tr>
<tr><td colspan="3">FA1</td></tr>
</table>

1- Selecciona la opción “Cancelar”.	2- Va a la ventana principal.
Prototipo de Interfaz	
Poscondiciones	Quedan registrados los códigos de barra de personas o vehículos en el lector Optimus S.

Tabla 2. 5 Descripción textual del caso de uso “Actualizar Lector”

Caso de Uso:	**Actualizar Base de Datos Local.**
Actores:	Técnico de Seguridad y protección (inicia)
Resumen:	El caso de uso inicia cuando el Técnico SP solicita actualizar la Base de Datos local, donde se registran las entradas o salidas según la causa. Finalizando así el caso de uso.
Precondiciones:	El Scanner debe ser conectado a un terminal de PC que contenga la BD local de personas, vehículos, y que contenga la aplicación. Debe existir una aplicación encargada de bajar información del lector portátil.
Referencias	**RF 10**
Prioridad	Alta
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El caso de uso inicia cuando el Técnico SP selecciona en la ventana principal la opción “Actualizar Base de Datos local”.	
	2- Muestra la interfaz que contiene las opciones “Entrada” y “Salida”. Y dos listados (personas o vehiculos) para seleccionar la causa del acceso.
3- Selecciona o introduce la causa del acceso. 4- Selecciona el tipo de acceso.	
	5- Guarda en la Base de Datos el tipo de acceso y la fecha en que ocurrió. Finalizando así el caso de uso.
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
Prototipo de Interfaz	
Poscondiciones	Queda registrada la entrada o salida de la persona o vehículo junto con la fecha y hora en la BD local.

Tabla 2. 6 Descripción textual del caso de uso “Actualizar Base de Datos Local”.

2.4.1.2-Vista Lógica

Esta vista representa un subconjunto del artefacto Modelo de Diseño, representando los elementos de diseño más importantes para la arquitectura del sistema. En esta vista se describe las clases más importantes, su organización en paquetes y subsistemas. También describe las realizaciones de casos de uso más importantes como por ejemplo las que describen aspectos dinámicos del sistema. Ver figura 6

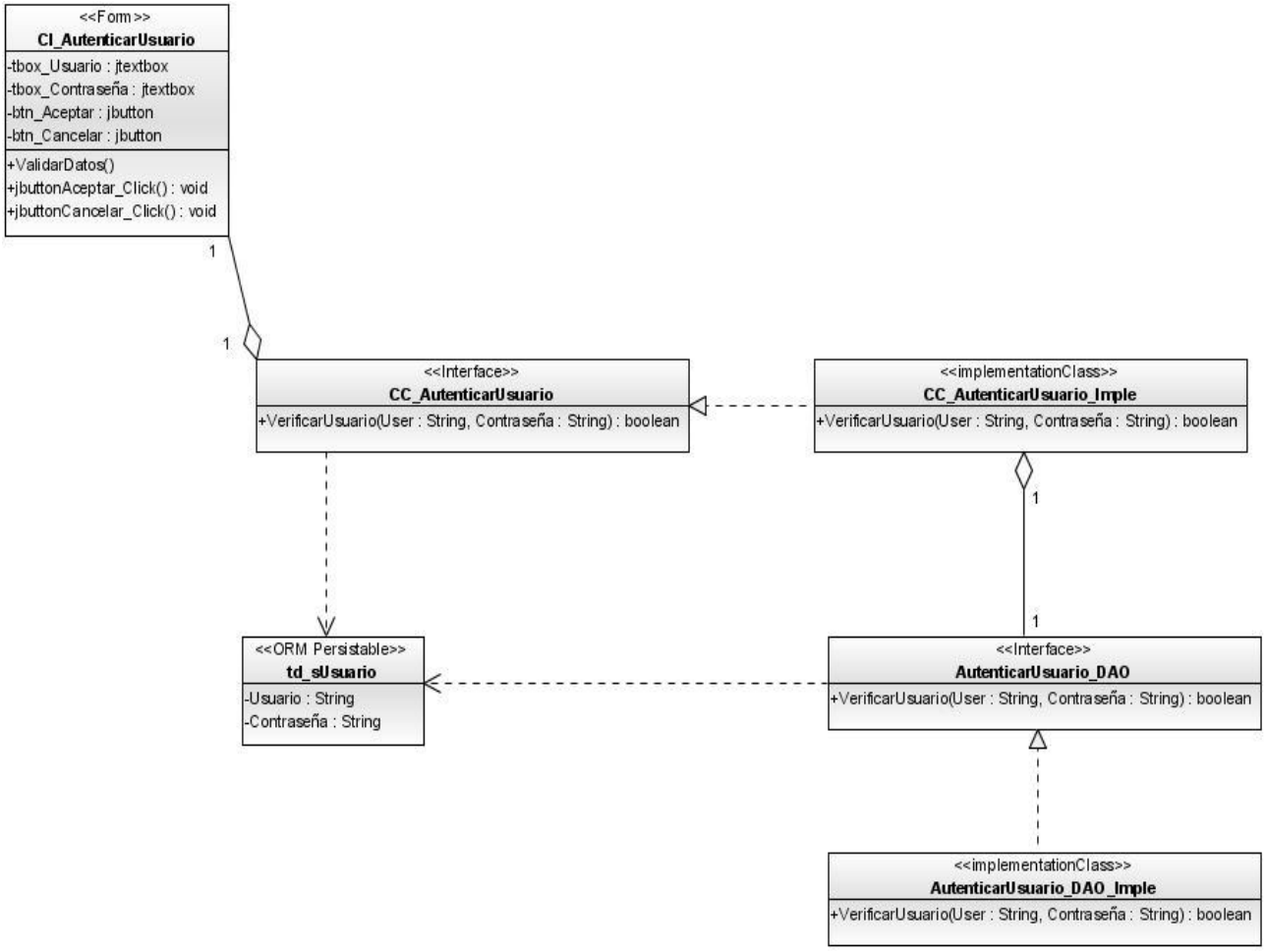


Figura 6 Modelo de Diseño.

2.4.1.3-Vista de Implementación

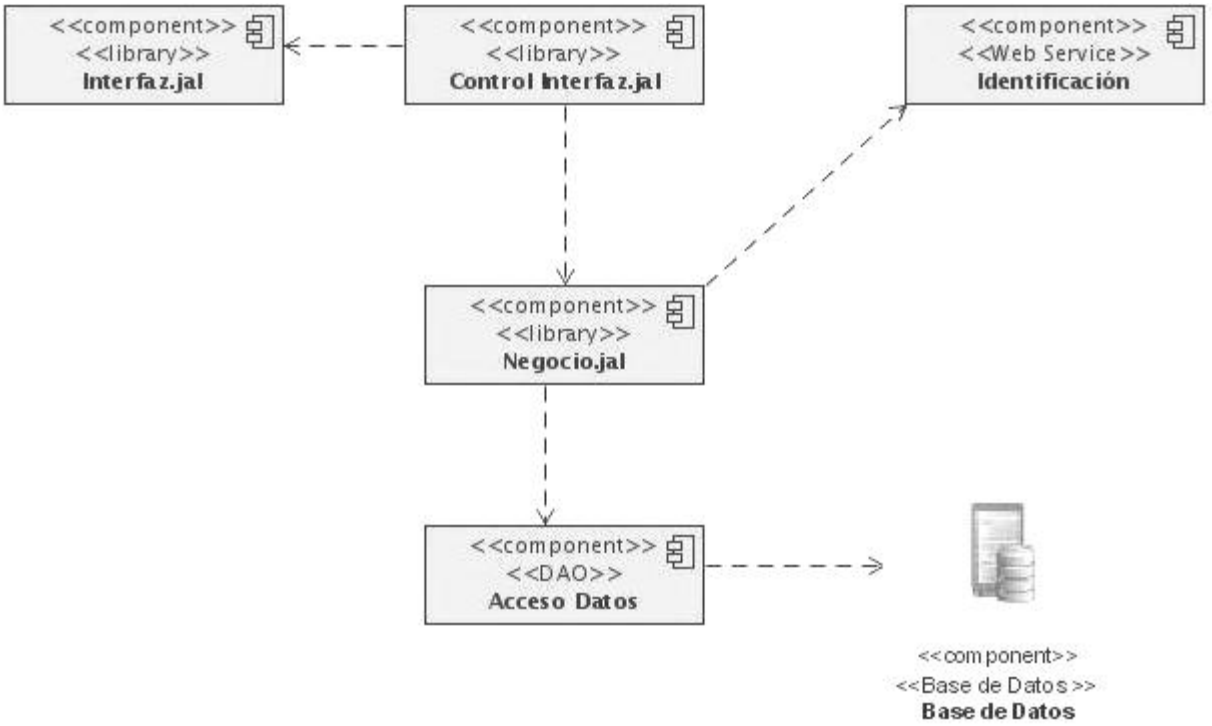


Figura 7 Paquete de componentes por capa.

Componente **Interfaz.jal:** Es donde se encuentran los formularios, es la encargada de recibir la entrada de datos por parte del usuario, se encargan de mostrar las funcionalidades al usuario y guiarlo en sus decisiones.

Componente **Control Interfaz.jal:** Están las clases donde se implementan las funcionalidades de la interfaz.

Componente **Negocio.jal:** Contendrá las clases del diseño que se encargan de gestionar toda la lógica del negocio.

Componente **Web Service Identificación:** A través de el se comunicará la aplicación con la base de datos del sistema para guardar y obtener informaciones del negocio.

Se utiliza el patrón DAO para abstraer y encapsular los accesos, para gestionar las conexiones a la fuente de datos y por ultimo obtener los datos almacenados.

Data Access Object (DAO, Objeto de Acceso a Datos) es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una base de datos o un archivo.

Las interfaces de los DAOs tendrán básicamente los siguientes métodos:

- **Métodos para descubrir:** Estos localizan los objetos almacenados para ser usados por la capa de servicios de negocio.

- **Métodos para persistir o salvar:** Estos hacen persistentes a los objetos transitorios.

- **Métodos para eliminar:** Estos eliminan a los objetos guardados en el medio de almacenamiento (base de datos).

- **Métodos para conteos y otras funciones agregadas:** Estos devuelven los resultados de operaciones que son más eficientes implementarlas usando funcionalidades de la base de datos (procedimientos almacenas, etcétera.) que iterar sobre los mismos objetos.

2.4.1.4-Vista de Despliegue

Mediante esta vista se obtiene una base para la comprensión de la distribución física de un sistema a través de nodos. Suele usarse cuando el sistema esta distribuido, y hay una traza directa del modelo de implementación puesto que cada componente físico debe

estar almacenado en un nodo, esto incluye también la asignación de tareas provenientes de la vista de procesos en los nodos. Ver figura 8

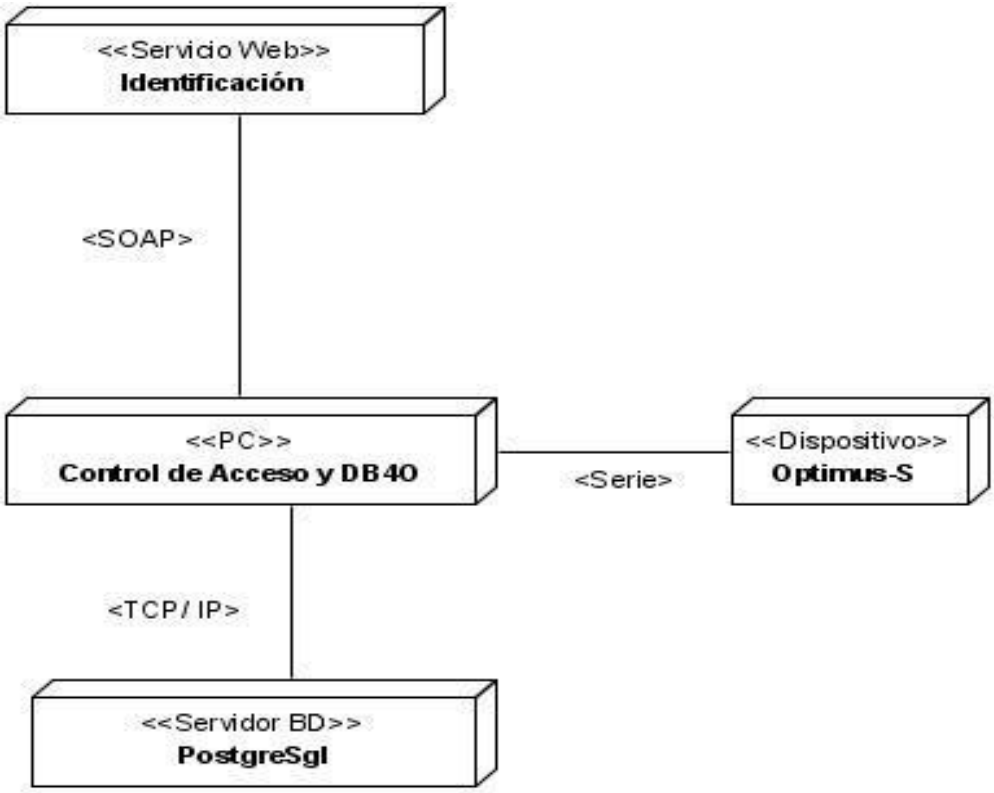


Figura 8 Diagrama de Despliegue.

Servicio Web Identificación: Este nodo representa al Servicio Web de Identificación el cual nos posibilita una serie de funcionalidades coherente a las credenciales de las personas de la Universidad de las Ciencias Informáticas.

PC Especialista: Este nodo representa la PC donde el especialista trabajará con la aplicación y el mismo contiene la aplicación y la base de dato local.

Servidor de Base de Datos: Este nodo contiene la base de dato principal de la aplicación.

Dispositivo Optimus-S: Este nodo representa al dispositivo Optimus-S el cual será de gran utilidad para agilizar el trabajo con la aplicación.

Descripción de elementos e interfaces de comunicación.

<<SOAP>>: Este es el protocolo usado para el intercambio de información cuando se consume el servicio web. Este es empleado entre la comunicación entre el servicio web Identificación y la aplicación.

<<Puerto de Serie>>: Este es una interfaz de comunicaciones de datos digitales utilizada frecuentemente por computadoras o periféricos, donde la información es transmitida bit a bit enviando uno solo a la vez. Este es empleado entre la comunicación de la aplicación y el dispositivo Optimus-S.

<<TCP/ IP>>: Es la base de Internet para que sirve para enlazar computadoras con diferentes sistemas operativos y proporciona transmisiones fiables de datos sobre la red. Este es empleado entre la comunicación del nodo que tiene la base de dato principal de la aplicación y el nodo que tiene la base de dato local.

2.4.1.5 -Vista de Procesos

Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.
En este caso la solución propuesta no presenta procesos concurrentes por tanto no se hace representación de esta vista.

2.5- Convenciones de nombres y estándares código

2.5.1 - Estándares de código en Java

Para el desarrollo de la aplicación se especifican estándares de desarrollo para los programadores en lenguaje Java, definiendo el modo en que básicamente debe ser realizada dichas aplicaciones.

1- Las líneas de código no deben ser muy extensas, por lo general inferiores a 80 caracteres. En caso de necesitar más, se bajan a la siguiente línea. Se puede aprovechar un cierre de paréntesis o una coma etc.

2- Se deben emplear espacios o tabuladores para que se distingan con claridad los distintos bloques de sentencias. Es recomendable emplear espacios en blanco debido a las distintas interpretaciones que hacen los editores sobre las tabulaciones.

3- Los nombres de las clases comienzan con mayúscula. Si el nombre es compuesto de varias palabras, por lo general se escriben juntas y con el primer carácter en mayúscula.

Ejemplo: class MyClase {…}

4- Las instancias de las clases, los objetos, comienzan con minúscula y en caso de estar compuesto de varias palabras, el resto comenzaran con mayúscula:

Ejemplo: public String myCadena;

5- El nombre de las variables y clases debe ser significativo, permitiendo tener una idea de su función con solo leerlo. Esto no siempre es posible, pero siempre es mejor dar un nombre cercano al objetivo que un nombre al azar.

6- Las variables siempre se deben inicializar si no queremos llevarnos sorpresas y excepciones.

7- No se puede emplear clases declaradas dentro de otra clase.

8- El estilo de los comentarios debe ser como el estilo de comentarios para C (***/* */*** ó ***//***).

9- Es muy recomendable documentar cada uno de los métodos y variables relevantes de la clase.

10- Es obligatorio detallar el uso de cada constante.

11- El comentario de la clase en general debería tener los siguientes campos:

- Nombre del autor.
- Fecha de creación.
- Descripción breve de sus funciones.
- Licencia bajo la que se ampara.
- Historial de cambios, en los que se incluiría el nombre del autor de los cambios, la fecha y un comentario sobre los mismos.

Los comentarios de bloques y líneas sueltas, se debe ser preciso, y no sobre comentar el código.

2.5.2 - Clases de la capa de acceso a Datos

Con el fin de lograr un lenguaje común y uniforme para las aplicaciones que se desarrollen, se especifican conversiones de nombres y estándares de código para los distintos recursos de las aplicaciones.

Las interfaces que enmarcan las operaciones sobre los objetos de acceso a datos, correspondientes al patrón de diseño Data Access Object terminan con la palabra *DAO.*

Ejemplo: *PersonaDAO*

Las implementaciones reales de las interfaces DAO comienza con el nombre de la interfaz correspondiente y termina con la palabra *Impl.*

Ejemplo: PersonaDAOImpl

Las clases utilizadas para efectuar pruebas a las interfaces DAO comienzan con el nombre de la interfaz correspondiente y termina con la palabra *Prueb*.

Ejemplo: *PersonaDAOPrueb*

2.5.3 - Clases de la capa de Servicios de Negocio

Las interfaces que representan las operaciones del negocio terminarán con la palabra *Negocio*.

Ejemplo: *PersonaNegocio*.

Las implementaciones de las interfaces de negocio comenzarán con el nombre de la interfaz correspondiente y terminaran con la palabra *Impl*.

Ejemplo: *PersonaNegocioImpl*

Las clases utilizadas para probar las funcionalidades del negocio terminan con la palabra *Prueb*.

Ejemplo: PersonaNegocioPrueb

2.5.4 - Recursos de la Capa de Presentación

- La Form principal se llamará "principal.java".
- Las clases asociadas a formularios se nombrarán correspondiéndose con la acción respectiva que realiza y terminarán con la extensión ".java".
- Las extensiones de las imágenes a utilizar vienen dadas por el formato de las mismas.
- Las extensiones de las imágenes a utilizar vienen dadas por el formato de las mismas.

2.8 - Conclusiones

En este capítulo se desarrollaron los modelos, que según RUP, son más importantes para la descripción de la arquitectura. Además se explicó como se aplica el patrón de arquitectura en capas y se definieron los estándares de código por capa.

Finalmente quedó establecida la arquitectura del Sistema de Control de Acceso de Personas y Vehículos de la Universidad de las Ciencias Informáticas.

CAPÍTULO 3: EVALUACIÓN DE LA ARQUITECTURA

3.1- Introducción

En el presente capítulo se realiza un análisis sobre el por qué, cuándo y cómo se evalúa la arquitectura, así como quiénes participan y qué se evalúa. Posteriormente es necesario hacer una valoración de la arquitectura propuesta en la investigación.

3.2- Evaluando una arquitectura de software

La calidad de un sistema depende en gran medida de su arquitectura. La evaluación de la misma permite obtener una visión de las decisiones más riesgosas asociadas al software en cuestión y además, aquellas que son buenas y correctas. Mientras mas temprano se mitigue un riesgo o se corrija un error, mejor es, tanto para el equipo de desarrollo como para el cliente.

“Cuanto más temprano se encuentre un problema en un proyecto de software, mejor. El costo de arreglar un error durante las fases de requerimientos o diseño, es mucho menor el costo de arreglar ese mismo error en la fase de verificación. Dado que la arquitectura es un producto temprano de la fase de diseño, esta tiene un profundo efecto en el sistema y en el proyecto.

Una mala arquitectura puede llevar a un proyecto al fracaso. Todos los requerimientos de calidad pueden quedar insatisfechos.

La arquitectura también determina la estructura del proyecto: configuración, agenda y presupuesto, alcance, entre otros aspectos. Es mejor cambiar la arquitectura antes que otros artefactos, que están basados en ella, se estabilicen.

Realizar una evaluación de la arquitectura es la manera más económica de evitar desastres.

3.2.1- ¿Cuándo una arquitectura puede ser evaluada?

Habitualmente, la evaluación de la arquitectura ocurre después que ésta ha sido especificada, pero antes que empiece la implementación, esto se hace con el objetivo de validar el diseño propuesto. No obstante, uno de los aspectos más interesantes de la evaluación de arquitecturas es que se puede efectuar en cualquier etapa de la vida de una arquitectura. Básicamente existen dos etapas: temprana y tardía. En la primera no tiene por qué estar especificada completamente la arquitectura, en la mayoría de los casos es utilizada para examinar las decisiones arquitectónicas ya tomadas y decidir entre las opciones que están pendientes. Por otro lado la evaluación tardía es realizada tanto cuando la arquitectura está terminada como cuando la implementación está completa.

“En general, una evaluación debe realizarse cuando hay suficiente de la arquitectura como para justificarlo. Una buena regla sería: *realizar una evaluación cuando el equipo de desarrollo empieza a tomar decisiones que dependen de la arquitectura y el costo de deshacerlas sobrepasa al costo de realizar una evaluación*.” (30)

3.2.2 - ¿Quiénes estan involucrados?

Durante la evaluación de la arquitectura participan los *stakeholders*: personal interesado en el desarrollo de la arquitectura y del software, pueden ser desarrolladores como el arquitecto, los diseñadores, el líder del proyecto, los implementadores o los verificadores, de ellos los tres primeros son *dicision markers* (tomadores de decisiones); por otro lado se tiene al equipo de evaluación, que no son más que personas que guiarán el proceso de evaluación y además realizarán el análisis. (30)

3.2.3 - Planificación de las evaluaciones

Las evaluaciones de la arquitectura pueden planearse o no. Una evaluación planeada es aquella que fue concebida dentro del ciclo de desarrollo, en cambio una no planeada ocurre cuando se han descubierto defectos en la arquitectura, por lo general en etapas tardías. Existen grandes riesgos de tener retrasos en la fecha de entrega como consecuencia de realizar una evaluación no planificada, es por ello que se recomienda planificar una o dos evaluaciones.

3.2.4 - ¿Qué resultados produce la evaluación de una arquitectura?

La evaluación de una arquitectura no produce resultados cuantitativos. El objetivo de mitigar los riesgos, es aprender cómo un atributo de calidad es afectado por una decisión de diseño arquitectónico, para que de esta manera se pueda estudiar con cuidado dicha decisión. La evaluación ayuda a encontrar debilidades, no dirá “correcto” o “incorrecto”, dirá donde se encuentran los riesgos. En caso de que existan contradicciones, el líder del proyecto deberá tomar la decisión. (30)

“En términos concretos, la evaluación de la arquitectura produce un informe, la forma y contenido del mismo varía según el método utilizado. En particular, produce repuestas a dos tipos de preguntas:

¿Es esta arquitectura adecuada para el sistema para la cual fue diseñada?

¿Cuál de dos o más arquitecturas propuestas es la más adecuada para el sistema?“

3.2.5 - ¿Cuáles son los beneficios de realizar una evaluación arquitectónica?

- Reúne a los stakeholders
- Fuerza una articulación en las metas específicas de calidad
- Fuerza una explicación clara de la arquitectura (30)

3.2.6 - Técnicas de evaluación

Existen varias técnicas de evaluación de la arquitectura de un software, básicamente se dividen en dos grupos: cualitativas y cuantitativas, ellas se utilizan a consecuencia del estado actual de la arquitectura. Cuando la arquitectura del sistema ha sido implantada se ponen en práctica técnicas cuantitativas, más cuando está en proceso de construcción se aplican técnicas cualitativas para su evaluación.

Dentro de las técnicas cualitativas se hallan cuestionarios, listas de verificación o escenarios. Por su parte las métricas, simulaciones, prototipos, experimentos y modelos matemáticos se clasifican como técnicas cuantitativas.

3.2.7- ¿Cómo se puede realizar una evaluación arquitectónica?

- Una de las características fundamentales de la tecnología RUP es que el sistema es centrado en la arquitectura, por lo que la calidad del sistema depende en gran medida de ella. Es posible estimar la calidad de la arquitectura a través de los requisitos no funcionales. Para ello se utilizará el método ARID (*Active Reviews Intermediate Designs*). Este método es el más conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Los roles involucrados en dicho método son:

  - Equipo de verificación.
  - Arquitecto.
  - Stakeholders.

  Cuenta con 9 pasos separados en dos fases:

  1-Pre-reunión

  - Identificar los revisores.
  - Preparar la presentación de diseño.
  - Preparar los escenarios.
  - Preparar los materiales.

  2-Evaluación

  - Presentación del ARID
  - Presentación del diseño.
  - Lluvia de ideas y priorización de escenarios.
  - Se realiza la revisión.
  - Conclusiones.(30)

3.3 - Evaluando la arquitectura propuesta

Dado el estado actual de la arquitectura y a pesar de que la propuesta ya fue validada por la dirección de informatización satisfactoriamente, se aplica la técnica cualitativa de evaluación por escenarios, específicamente el método ARID. “El método ARID ya que realiza la evaluación de diseños parciales en las etapas tempranas de desarrollo.”
A continuación se definen los escenarios fundamentales de la arquitectura del sistema:

- **Rendimiento**

Un buen rendimiento supone obtener buenos y esperados resultados, nuestro sistema presenta una interfaz amigable para su interacción con el agente de seguridad y protección, las herramientas y tecnologías propuestas todas de bajo licencias libres su resultado final no será otro que garantizar el control estricto de los accesos, tanto de personas como vehículos garantizando así la seguridad y protección física de los medios existentes.

- **Seguridad**

Sin dudas la seguridad es uno de los elementos más primordiales de cualquier arquitectura, y por supuesto no es pasada por alto. La solución propuesta se basa principalmente en el empleo autenticación como una acción de obligatorio cumplimiento, La contraseña pasa por un proceso de encriptación siguiendo el algoritmo RSA. Este método de encriptación también se usará para encriptar los mensajes que se envíen.

- **Portabilidad**

El sistema utiliza la tecnología Java2EE, emplea PostgreSQL como gestor de base de datos relacionales, además db4o como base de dato embebida, Visual Paradigm como herramienta de modelado y NetBeans como ambiente de desarrollo integrado. Todo lo antes mencionado está liberado por licencias libres, es decir son tecnologías “open Source” lo cual implica que se puedan ejecuta en diferentes plataformas, fundamentalmente Windows, Unix, Linux entre otras.

- **Disponibilidad**

Nuestro sistema tiene la posibilidad mediante la base de datos local instalada en los puntos de entrada, almacenar una copia de la base de datos central lo que permitirá que en todo momento nuestro sistema este disponible para los usuarios y en caso de fallar la conexión el sistema no se afecte, es decir el control de los accesos seguirá siendo controlado correctamente.

2.4 - Seguridad

La seguridad del sistema, se basa principalmente en el empleo de la autenticación como una acción de obligatorio cumplimiento para el posterior uso del mismo, los cuáles serán verificados antes de darle acceso a las funcionalidades de sistema y si alguno de estos datos son incorrectos, se denegará dicho acceso. La contraseña pasa por un proceso de encriptación siguiendo el algoritmo RSA el cual posee 2 llaves una pública y otra privada y un algoritmo matemático, este método de encriptación también se usará para encriptar los mensajes que se envíen.

RSA es el más conocido y usado de los sistemas de clave pública, y también el más rápido de ellos. Presenta todas las ventajas de los sistemas asimétricos, incluyendo la firma digital, aunque resulta más útil a la hora de implementar la confidencialidad el uso de sistemas simétricos, por ser más rápidos. Se suele usar también en los sistemas mixtos para encriptar y enviar la clave simétrica que se usará posteriormente en la comunicación cifrada.

El sistema RSA se basa en el hecho matemático de la dificultad de factorizar números muy grandes. Para factorizar un número el sistema más lógico consiste en empezar a dividir sucesivamente éste entre 2, entre 3, entre 4,..., y así sucesivamente, buscando que el resultado de la división sea exacto, es decir, de resto 0, con lo que ya tendremos un divisor del número.

El cálculo de estas claves se realiza en secreto en la máquina en la que se va a guardar la clave privada, y una vez generada ésta conviene protegerla mediante un algoritmo criptográfico simétrico. (29)

3.5 - Análisis Económico

CONCEPTO DE GASTO	GASTO	TOTAL X MES	TOTAL
Material Gastable, Material de oficina: • **Hojas** • **Lapiceros** • **Presilladora** • **Tonel**	$100.00	-	$100.00
Equipos Necesarios, Equipos de computación • **1 Impresora** • **1 Memoria flash** • **1 PC**	$1 00.00	-	$100.00
Gastos directos • **Energía eléctrica** • **Alimentos** • **Transporte**	$45.00 $30.00 $10.00 $15.00	$5.00 $6.00 - $3.00	$100.00

• **Teléfonos**			
COSTO TOTAL			$300.00

Tabla3. 1 Análisis del costo económico

Para dar comienzo a esta investigación científica se contaba con un presupuesto inicial de unos $570.00 pero luego de realizar el respectivo análisis financiero se llegó a la conclusión de que los gastos para el tiempo dedicado a la realización de la misma solo se alcanzó $300.00. Producto de esto queda evidenciado que fue una investigación bastante factible, superando las expectativas previstas.

3.4 - Conclusiones

La Universidad de la Ciencias Informáticas se encuentra inmersa en una estrategia de migración a software libre de todos los sistemas que se desarrollen. Con esta premisa, la prioridad fue lograr una arquitectura que cumpliera con los nuevos requerimientos, basada en tecnologías y licencias libres; logrando así una arquitectura robusta y flexible que asimilara los cambios con facilidad

CONCLUSIONES

Durante el desarrollo de la investigación se cumplieron las tareas y objetivos propuestos:

- Se propuso el diseño de la arquitectura para el Sistema de Control de Acceso de Personas y Vehículos de la Universidad de las Ciencias Informáticas, el cual permite el control de las personas y los vehículos que acceden a la universidad.
- Se realizó un análisis del sistema actual así como un estudio del arte referente a las arquitecturas de los softwares similares existentes asociados al campo de acción.
- Se evaluaron las metodologías, tecnologías y tendencias actuales, lo que permitió la selección de las herramientas para software libre que más se adecuan a las necesidades para el desarrollo del software.
- Se lograron otros objetivos como la definición de los estándares de código a utilizar así como el análisis económico de la investigación.

RECOMENDACIONES

Las metas planteadas en este trabajo se han cumplido en base a los resultados obtenidos y de un conjunto de ideas surgidas a lo largo de la investigación que no están enmarcadas dentro de los objetivos de este trabajo, aunque sí forman parte del área de investigación, se recomienda:

- Continuar con el refinamiento constante de la arquitectura propuesta, durante el ciclo de desarrollo.
- Realizar una evaluación de los costes de las tecnologías utilizadas, aunque las mismas sean en su totalidad libre, es necesario evaluar los costes del hardware que la soportan.
- Seguir profundizando en cuanto al tema de la seguridad del Sistema.
- El sistema con la arquitectura que se propone exige para un correcto funcionamiento, recursos del hardware, a pesar de que la tecnología de desarrollo que se utilizarán en la arquitectura propuesta es libre, por lo que se recomienda obtener los equipos adecuados para lograr el mejor resultado de la propuesta que se brinda en la investigación.

BIBLIOGRAFÍA

1. Arquitectura. *http://es.wikipedia.org/wiki/Arquitectura_software#Arquitecturas_m.C3.A1s_comunes.* [Online] 2 20, 2008.

2. Introducción a la Arquitectura de Software. *http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx.* [Online] 2 20, 2008.

3. Control de Acceso. *http://www.itecmex.com/control.htm.* [Online] 2 20, 2008.

4. Software Control Biométrico de personal. *http://www.ddsoftware.com.ar/software-control-biometrico.html.* [Online] 1 4, 2008.

5. http://community.igalia.com/twiki/pub/Fisterra/ProjectArticles/fisterra-freesoft-project.pdf. *http://community.igalia.com/twiki/pub/Fisterra/ProjectArticles/fisterra-freesoft-project.pdf.* [Online] 2 9, 2008.

6. Lenguaje Unificado de Modelado. *http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado.* [Online] 2 3, 2008.

7. Que es Java. *http://java.ciberaula.com/articulo/que_es_java/.* [Online] 2 13, 2008.

8. .Net. *http://es.wikipedia.org/wiki/.NET.* [Online] 2 13, 2008.

9. Patrones Arquitectónicos. *http://www.lsi.us.es/docencia/get.php?id=1130.* [Online] 2 14, 2008.

10. Patrones Arquitectónicos. *http://www.lsi.us.es/docencia/get.php?id=1130.* [Online] 2 15, 2008.

11. Sistema Gestor de base de datos SGBD. *http://www.error500.net/garbagecollector/archives/categorias/bases_de_datos/sistema_gestor_de_base_de_datos_sgbd.php.* [Online] 3 2, 2088.

12. Sistema Gestor de Base de Datos PostgreSQL. *http://www.http-peru.com/postgresql.php.* [Online] 3 6, 2008.

13. Noticias. *http://www.mysql-hispano.org/.* [Online] 3 14, 2008.

14. DB4O. *http://es.wikipedia.org/wiki/DB4O.* [Online] 3 14, 2008.

15. Persistencia de Objetos Java utilizando db4o. *http://www.programacion.net/java/articulo/jap_persis_db4o/.* [Online] 3 15, 2008.

16. SOLite. *http://es.wikipedia.org/wiki/SQLite.* [Online] 3 16, 2008.

17. Entorno de desarrollo integrado. *http://es.wikipedia.org/wiki/Ambiente_integrado_de_desarrollo.* [Online] 3 16, 2008.

18. Eclipse - an open development platform. *http://www.eclipse.org/.* [Online] 3 17, 2008.

19. Aplicación para el desarrollo en java. *http://netbeans-ide.softonic.com/.* [Online] 3 17, 2008.

20. Introducción a Visual Studio. *http://msdn.microsoft.com/es-es/library/fx6bk1f4(VS.80).aspx.* [Online] 3 17, 2008.

21. Subversion. *http://es.wikipedia.org/wiki/Subversion.* [Online] 3 17, 2008.

22. HERRAMIENTAS . *www.eici.ucm.cl/Academicos/ygomez/descargas/Ing_Sw2/apuntes/HERRAMIENTAS - CASEyproductos.doc.* [Online] 3 18, 2008.

23. Noticias. *www.programacion.com/noticia/1363.* [Online] 3 18, 2008.

24. Rational Rose. *http://www.slideshare.net/vivi_jocadi/rational-rose/.* [Online] 3 18, 2008.

25. Rational Rose Enterprise. http://www.rational.com.ar/herramientas/roseenterprise.html. [Online] 4 2, 2008.

26. Framework. *http://es.wikipedia.org/wiki/Framework.* [Online] 4 2, 2008.

27. Hibernate. *http://mundogeek.net/archivos/2007/01/27/hibernate/.* [Online] 4 3, 2008.

28. db4o. *http://www.db4o.com/community/blogs/espanol/atom.xml.* [Online] 5 1, 2008.

29.Encriptacion. *http://www.textoscientificos.com/redes/redes-virtuales/tuneles/encriptacion.* [Online] 5 5, 2008.

30. Evaluación de la arquitectura de software.*www.fing.edu.uy.* [Online] 5 5, 2008.

GLOSARIO

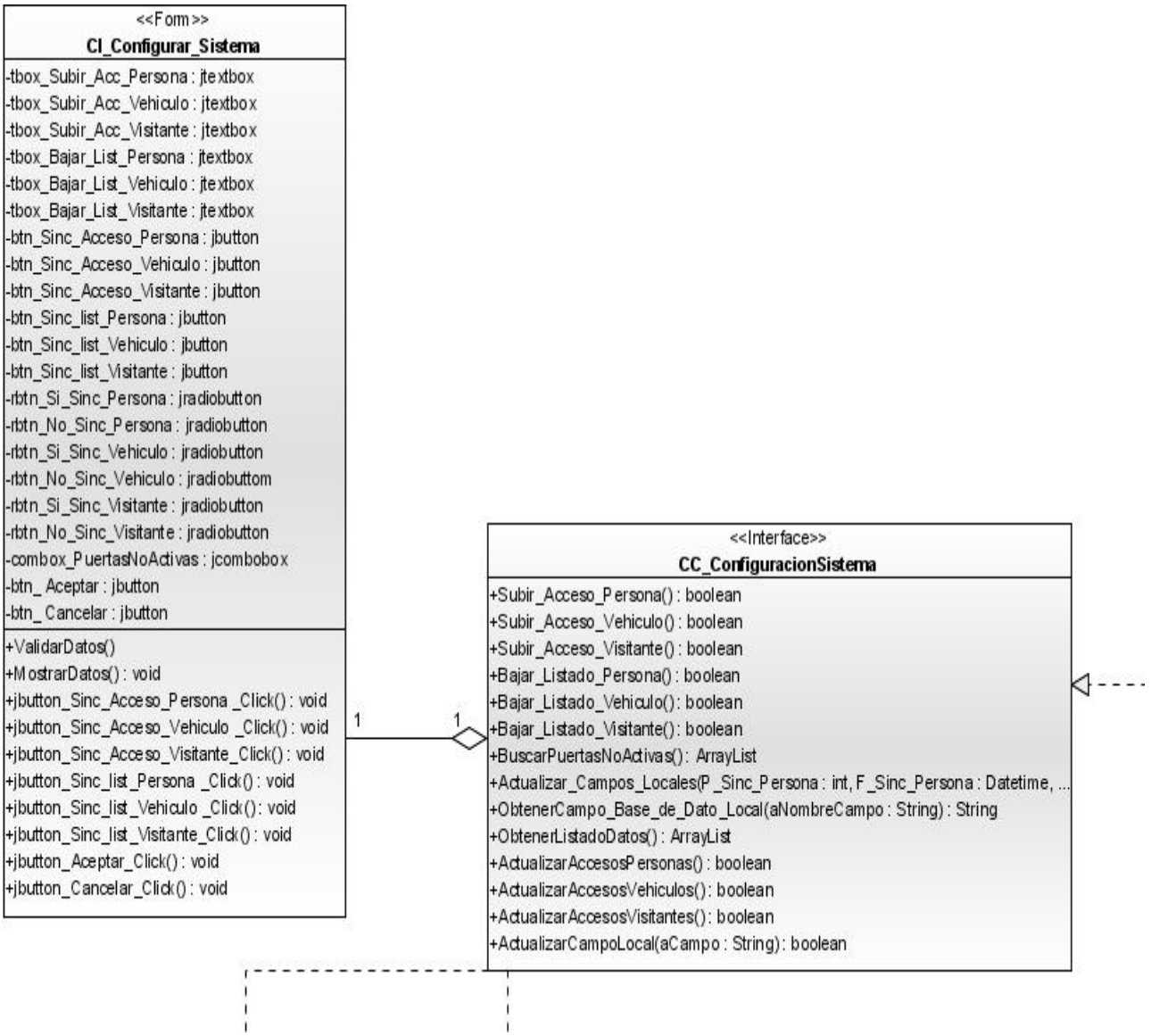
1-SQL: Es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas.

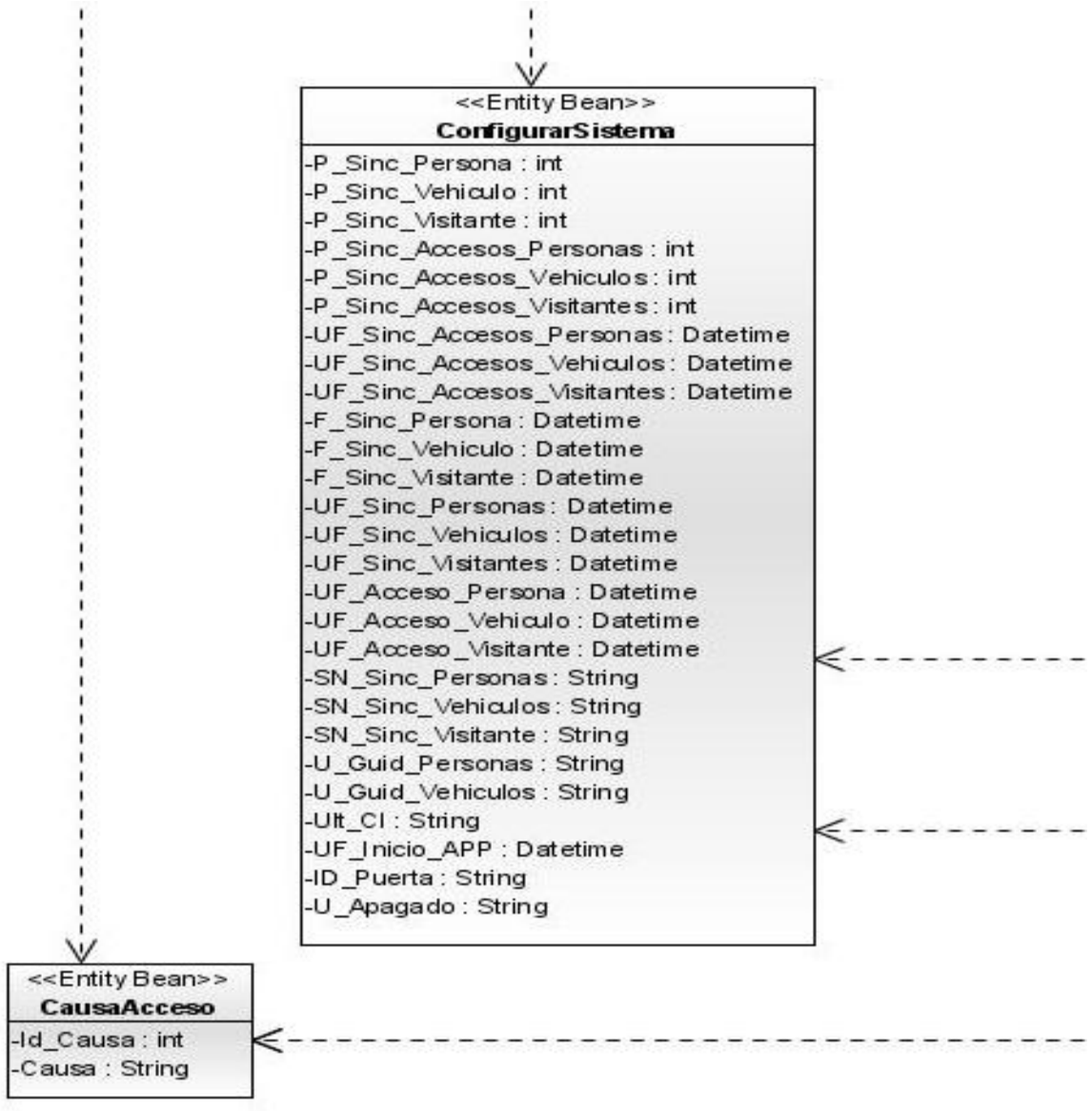
2-Smalltalk: Es un sistema informático que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales. Metafóricamente, se puede considerar que un Smalltalk es un mundo virtual donde viven objetos que se comunican mediante el envío de mensajes.

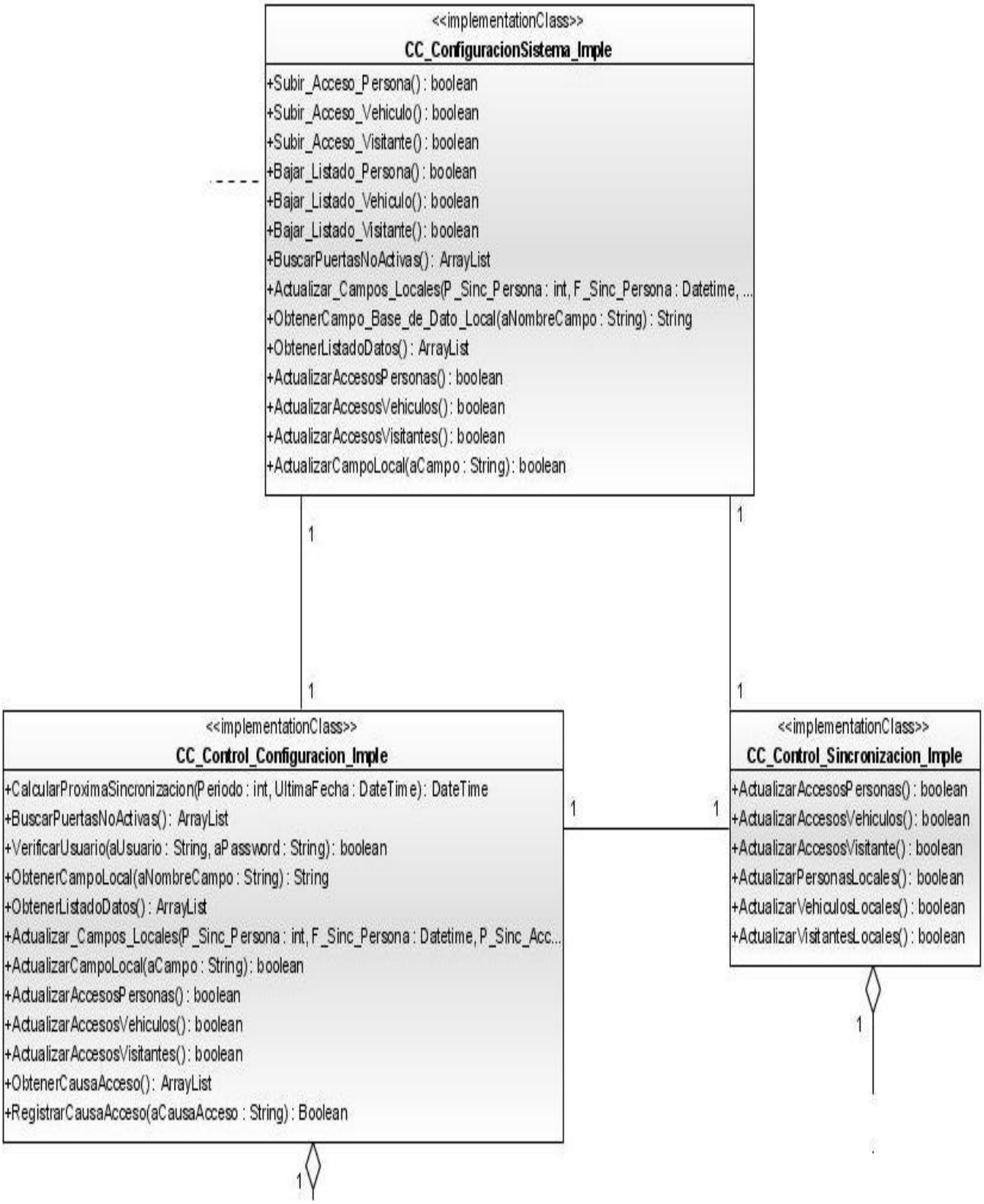
3-ACID: Se denomina ACID a la propiedad de una base de datos para realizar transacciones seguras.

4-CVS-El Concurrent Versions System (CVS), también conocido como Concurrent Versioning System, es una aplicación informática que implementa un sistema de control de versiones

ANEXOS


<<Form>>
CI_Configurar_Sistema
-tbox_Subir_Acc_Persona : jtextbox
-tbox_Subir_Acc_Vehiculo : jtextbox
-tbox_Subir_Acc_Visitante : jtextbox
-tbox_Bajar_List_Persona : jtextbox
-tbox_Bajar_List_Vehiculo : jtextbox
-tbox_Bajar_List_Visitante : jtextbox
-btn_Sinc_Acceso_Persona : jbutton
-btn_Sinc_Acceso_Vehiculo : jbutton
-btn_Sinc_Acceso_Visitante : jbutton
-btn_Sinc_list_Persona : jbutton
-btn_Sinc_list_Vehiculo : jbutton
-btn_Sinc_list_Visitante : jbutton
-rbtn_Si_Sinc_Persona : jradiobutton
-rbtn_No_Sinc_Persona : jradiobutton
-rbtn_Si_Sinc_Vehiculo : jradiobutton
-rbtn_No_Sinc_Vehiculo : jradiobuttom
-rbtn_Si_Sinc_Visitante : jradiobutton
-rbtn_No_Sinc_Visitante : jradiobutton
-combox_PuertasNoActivas : jcombobox
-btn_ Aceptar : jbutton
-btn_ Cancelar : jbutton
+ValidarDatos()
+MostrarDatos() : void
+jbutton_Sinc_Acceso_Persona _Click() : void
+jbutton_Sinc_Acceso_Vehiculo _Click() : void
+jbutton_Sinc_Acceso_Visitante_Click() : void
+jbutton_Sinc_list_Persona _Click() : void
+jbutton_Sinc_list_Vehiculo _Click() : void
+jbutton_Sinc_list_Visitante_Click() : void
+jbutton_Aceptar_Click() : void
+jbutton_Cancelar_Click() : void
1
1
<<Interface>>
CC_ConfiguracionSistema
+Subir_Acceso_Persona() : boolean
+Subir_Acceso_Vehiculo() : boolean
+Subir_Acceso_Visitante() : boolean
+Bajar_Listado_Persona() : boolean
+Bajar_Listado_Vehiculo() : boolean
+Bajar_Listado_Visitante() : boolean
+BuscarPuertasNoActivas() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, ...
+ObtenerCampo_Base_de_Dato_Local(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+ActualizarAccesosPersonas() : boolean
+ActualizarAccesosVehiculos() : boolean
+ActualizarAccesosVisitantes() : boolean
+ActualizarCampoLocal(aCampo : String) : boolean


<<Entity Bean>>
ConfigurarSistema
-P_Sinc_Persona : int
-P_Sinc_Vehiculo : int
-P_Sinc_Visitante : int
-P_Sinc_Accesos_Personas : int
-P_Sinc_Accesos_Vehiculos : int
-P_Sinc_Accesos_Visitantes : int
-UF_Sinc_Accesos_Personas : Datetime
-UF_Sinc_Accesos_Vehiculos : Datetime
-UF_Sinc_Accesos_Visitantes : Datetime
-F_Sinc_Persona : Datetime
-F_Sinc_Vehiculo : Datetime
-F_Sinc_Visitante : Datetime
-UF_Sinc_Personas : Datetime
-UF_Sinc_Vehiculos : Datetime
-UF_Sinc_Visitantes : Datetime
-UF_Acceso_Persona : Datetime
-UF_Acceso_Vehiculo : Datetime
-UF_Acceso_Visitante : Datetime
-SN_Sinc_Personas : String
-SN_Sinc_Vehiculos : String
-SN_Sinc_Visitante : String
-U_Guid_Personas : String
-U_Guid_Vehiculos : String
-Ult_CI : String
-UF_Inicio_APP : Datetime
-ID_Puerta : String
-U_Apagado : String
<<Entity Bean>>
CausaAcceso
-Id_Causa : int
-Causa : String


<<implementationClass>>
CC_ConfiguracionSistema_Imple
+Subir_Acceso_Persona() : boolean
+Subir_Acceso_Vehiculo() : boolean
+Subir_Acceso_Visitante() : boolean
+Bajar_Listado_Persona() : boolean
+Bajar_Listado_Vehiculo() : boolean
+Bajar_Listado_Visitante() : boolean
+BuscarPuertasNoActivas() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, ...
+ObtenerCampo_Base_de_Dato_Local(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+ActualizarAccesosPersonas() : boolean
+ActualizarAccesosVehiculos() : boolean
+ActualizarAccesosVisitantes() : boolean
+ActualizarCampoLocal(aCampo : String) : boolean
1
1
1
1
<<implementationClass>>
CC_Control_Configuracion_Imple
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime) : DateTime
+BuscarPuertasNoActivas() : ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String) : boolean
+ObtenerCampoLocal(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P_Sinc_Acc...
+ActualizarCampoLocal(aCampo : String) : boolean
+ActualizarAccesosPersonas() : boolean
+ActualizarAccesosVehiculos() : boolean
+ActualizarAccesosVisitantes() : boolean
+ObtenerCausaAcceso() : ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean
1
1
1
1
<<implementationClass>>
CC_Control_Sincronizacion_Imple
+ActualizarAccesosPersonas() : boolean
+ActualizarAccesosVehiculos() : boolean
+ActualizarAccesosVisitante() : boolean
+ActualizarPersonasLocales() : boolean
+ActualizarVehiculosLocales() : boolean
+ActualizarVisitantesLocales() : boolean
1

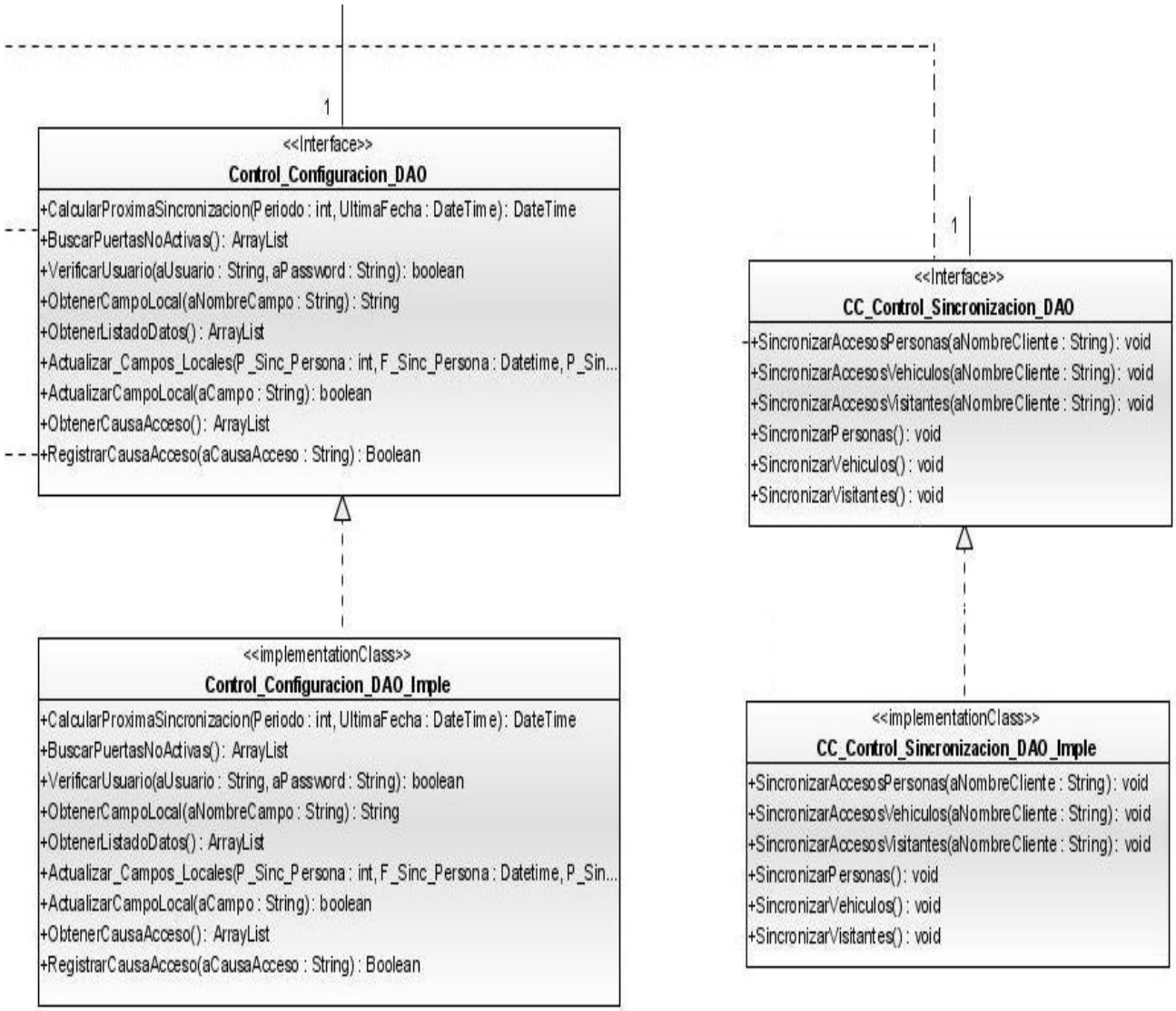
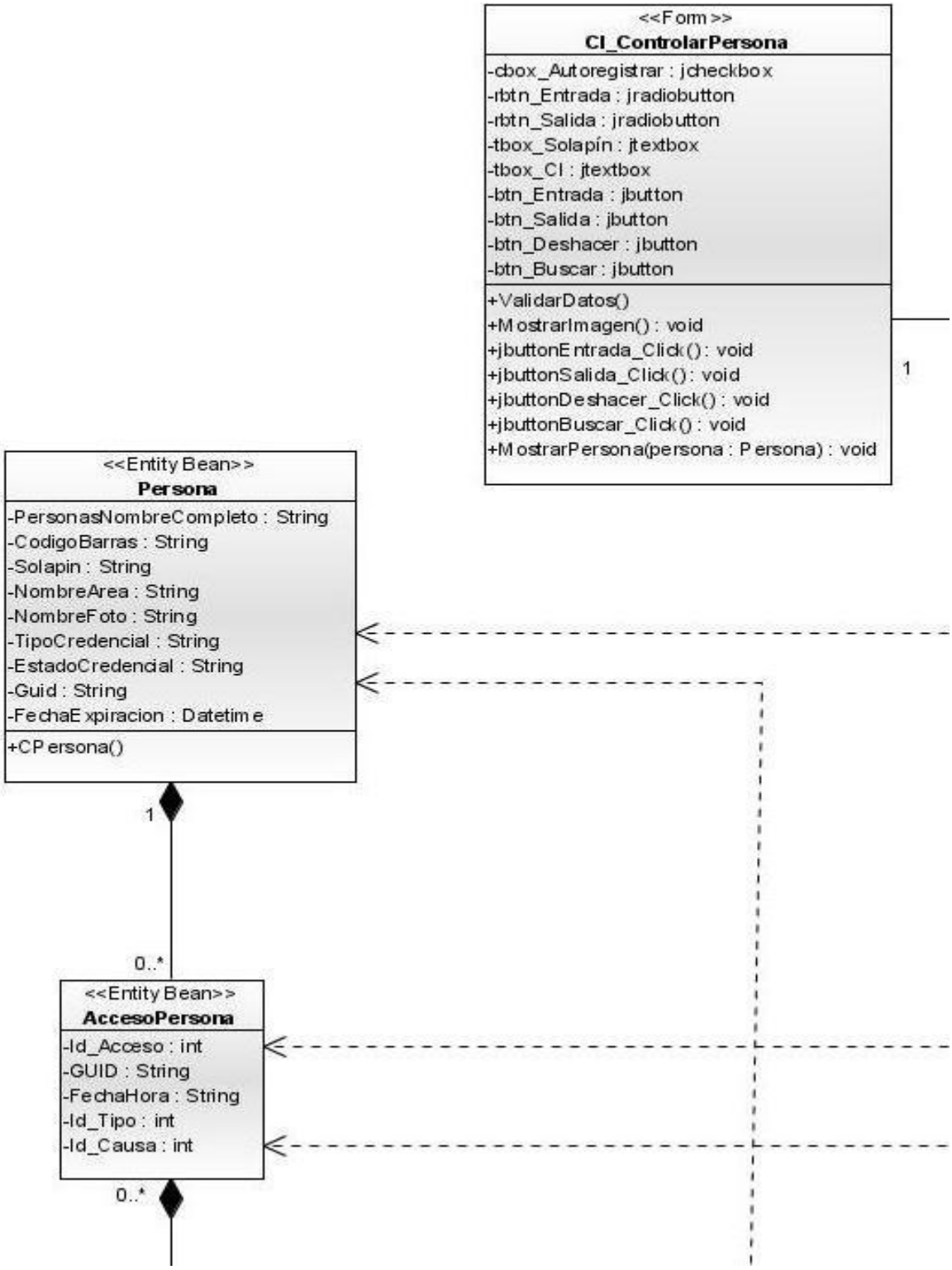
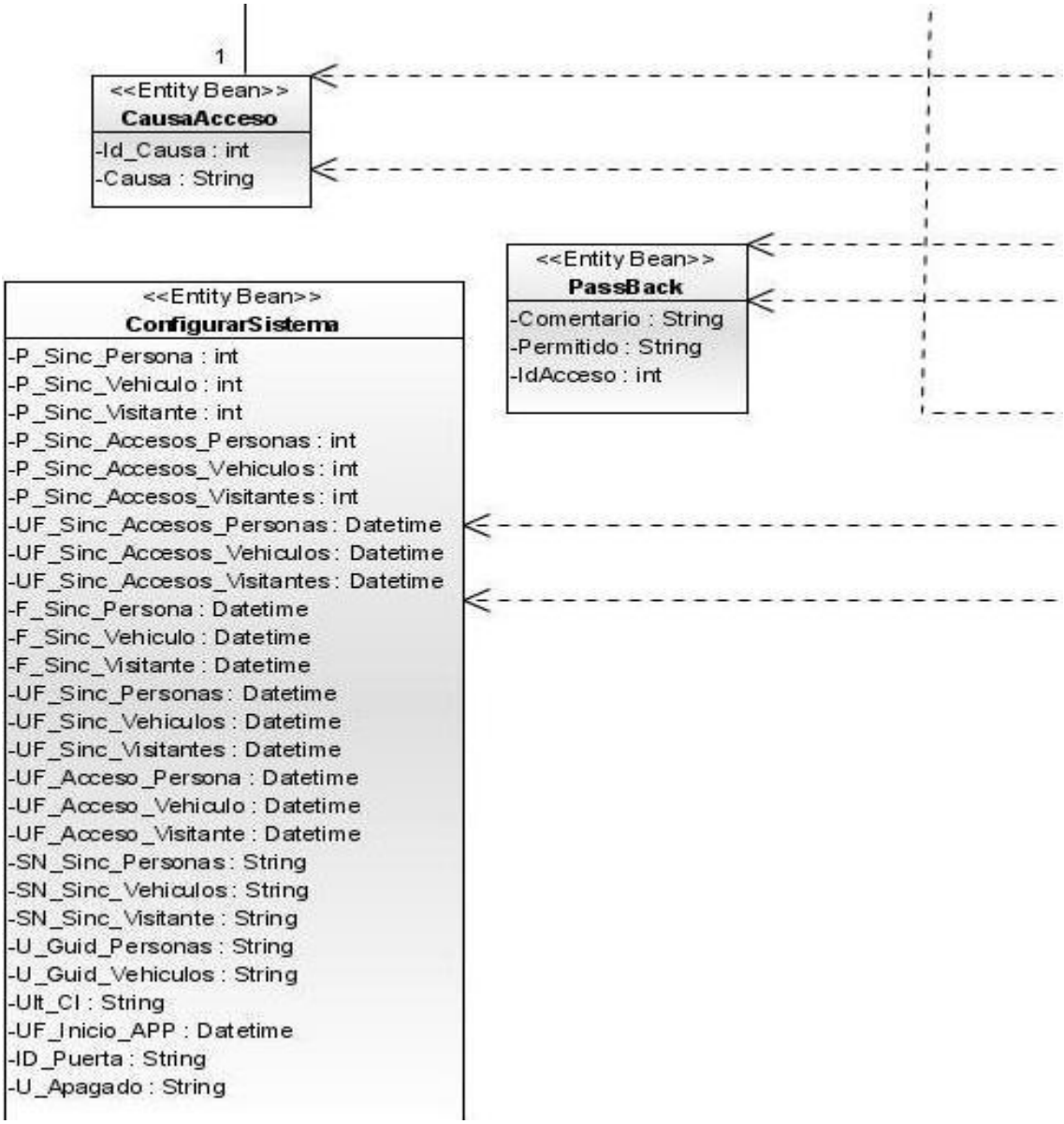
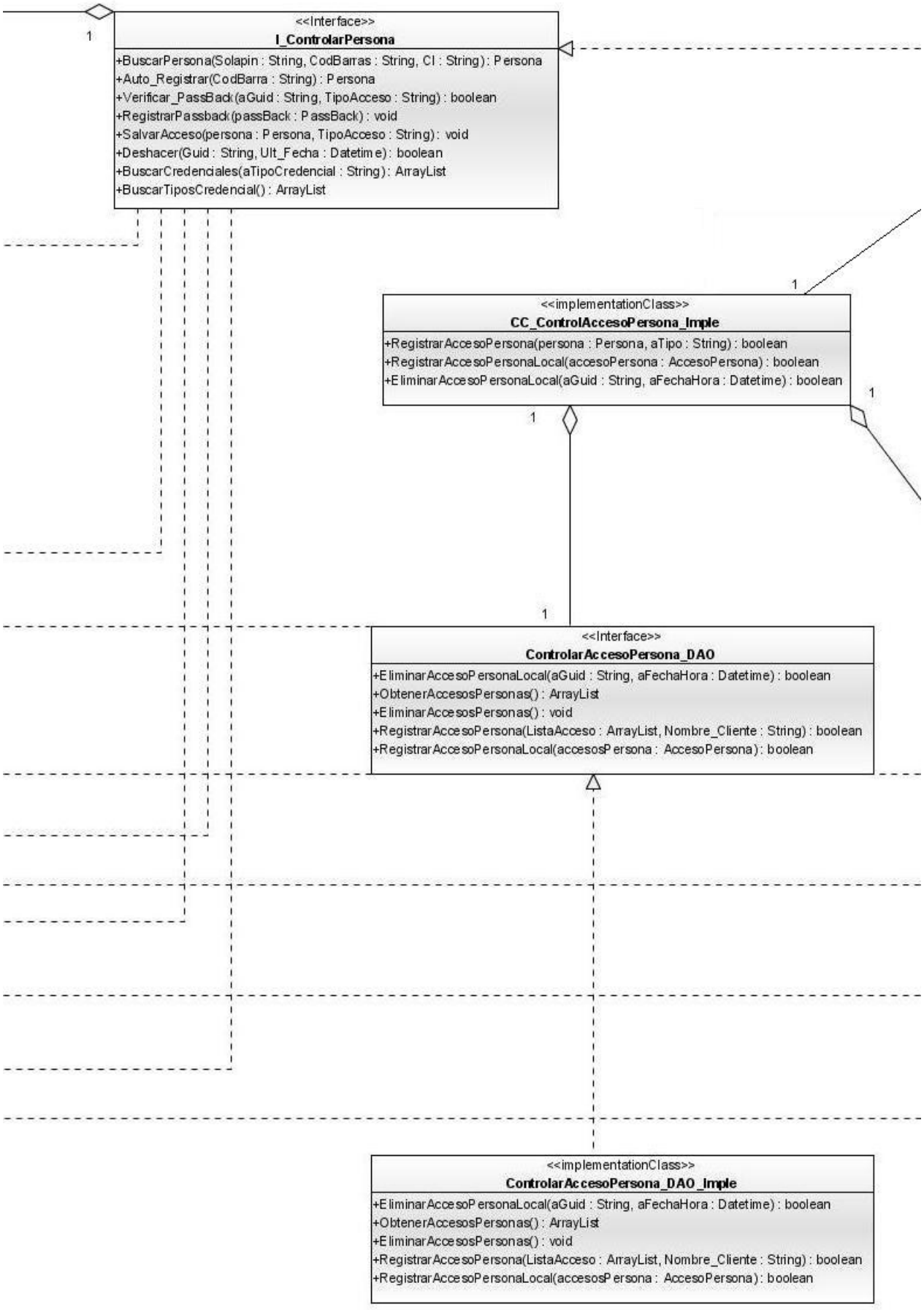
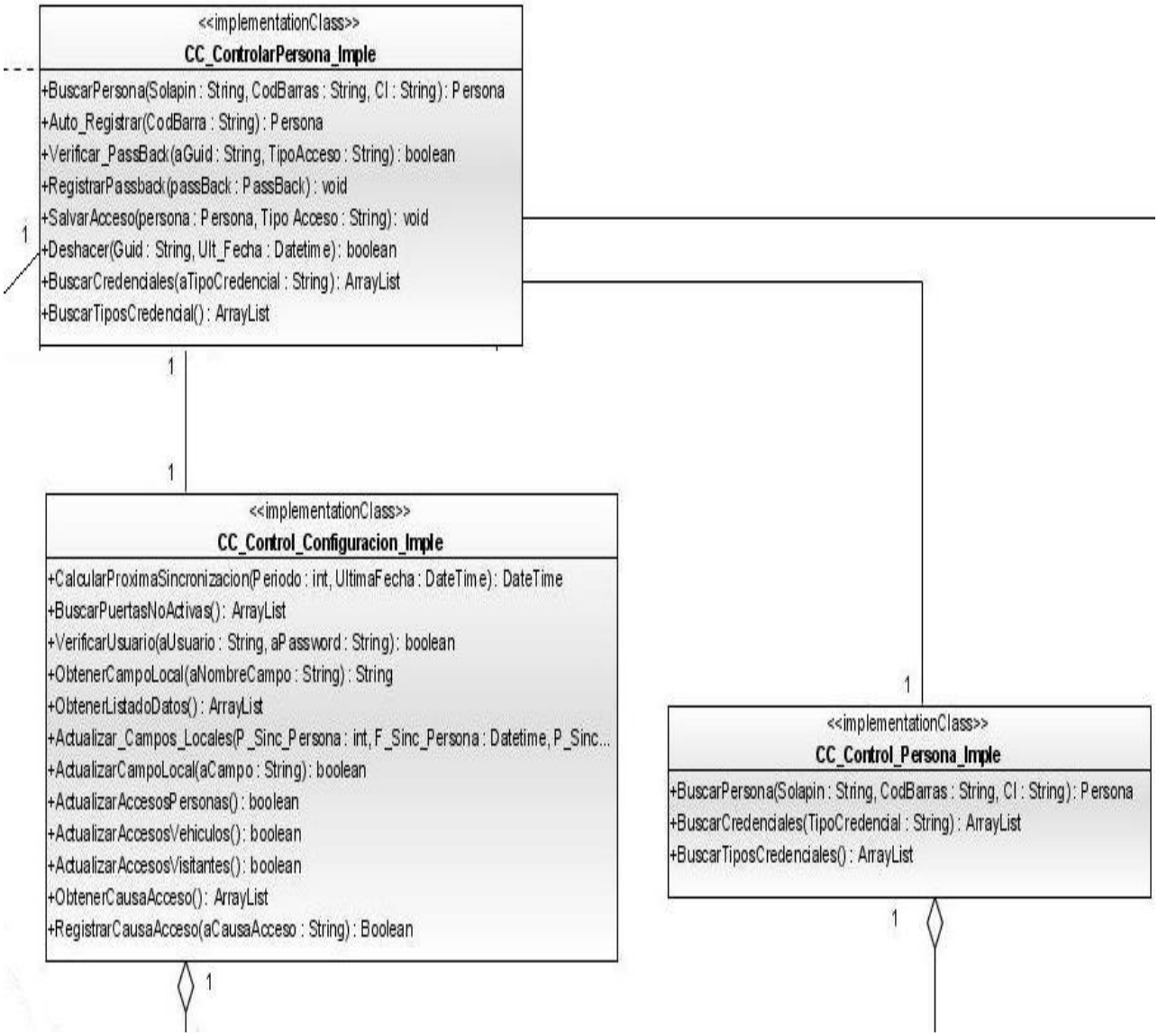


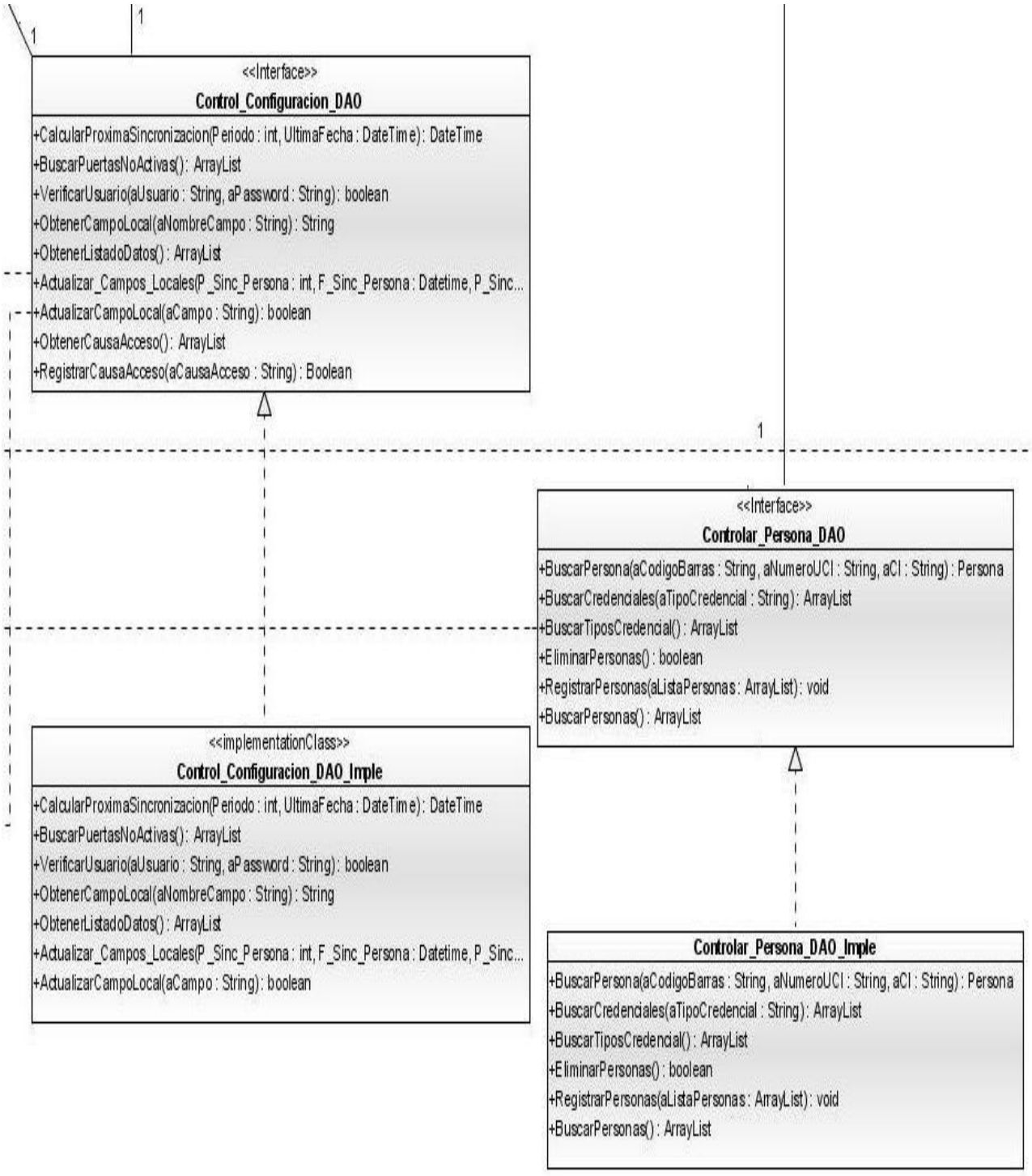
Figura 9 Diagrama de clase del diseño “Configurar Sistema”


<<Form>>
CI_ControlarPersona
-cbox_Autoregistrar : jcheckbox
-rbtn_Entrada : jradiobutton
-rbtn_Salida : jradiobutton
-tbox_Solapín : jtextbox
-tbox_CI : jtextbox
-btn_Entrada : jbutton
-btn_Salida : jbutton
-btn_Deshacer : jbutton
-btn_Buscar : jbutton
+ValidarDatos()
+MostrarImagen() : void
+jbuttonEntrada_Click() : void
+jbuttonSalida_Click() : void
+jbuttonDeshacer_Click() : void
+jbuttonBuscar_Click() : void
+MostrarPersona(persona : Persona) : void
1
<<Entity Bean>>
Persona
-PersonasNombreCompleto : String
-CodigoBarras : String
-Solapin : String
-NombreArea : String
-NombreFoto : String
-TipoCredencial : String
-EstadoCredencial : String
-Guid : String
-FechaExpiracion : Datetime
+CPersona()
1
0..*
<<Entity Bean>>
AccesoPersona
-Id_Acceso : int
-GUID : String
-FechaHora : String
-Id_Tipo : int
-Id_Causa : int
0..*


1
<<Entity Bean>>
CausaAcceso
-Id_Causa : int
-Causa : String
<<Entity Bean>>
PassBack
-Comentario : String
-Permitido : String
-IdAcceso : int
<<Entity Bean>>
ConfigurarSistema
-P_Sinc_Persona : int
-P_Sinc_Vehiculo : int
-P_Sinc_Visitante : int
-P_Sinc_Accesos_Personas : int
-P_Sinc_Accesos_Vehiculos : int
-P_Sinc_Accesos_Visitantes : int
-UF_Sinc_Accesos_Personas : Datetime
-UF_Sinc_Accesos_Vehiculos : Datetime
-UF_Sinc_Accesos_Visitantes : Datetime
-F_Sinc_Persona : Datetime
-F_Sinc_Vehiculo : Datetime
-F_Sinc_Visitante : Datetime
-UF_Sinc_Personas : Datetime
-UF_Sinc_Vehiculos : Datetime
-UF_Sinc_Visitantes : Datetime
-UF_Acceso_Persona : Datetime
-UF_Acceso_Vehiculo : Datetime
-UF_Acceso_Visitante : Datetime
-SN_Sinc_Personas : String
-SN_Sinc_Vehiculos : String
-SN_Sinc_Visitante : String
-U_Guid_Personas : String
-U_Guid_Vehiculos : String
-Ult_CI : String
-UF_Inicio_APP : Datetime
-ID_Puerta : String
-U_Apagado : String


1
<<Interface>>
I_ControlarPersona
+BuscarPersona(Solapin : String, CodBarras : String, CI : String) : Persona
+Auto_Registrar(CodBarra : String) : Persona
+Verificar_PassBack(aGuid : String, TipoAcceso : String) : boolean
+RegistrarPassback(passBack : PassBack) : void
+SalvarAcceso(persona : Persona, TipoAcceso : String) : void
+Deshacer(Guid : String, Ult_Fecha : Datetime) : boolean
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarTiposCredencial() : ArrayList
1
<<implementationClass>>
CC_ControlAccesoPersona_Imple
+RegistrarAccesoPersona(persona : Persona, aTipo : String) : boolean
+RegistrarAccesoPersonaLocal(accesoPersona : AccesoPersona) : boolean
+EliminarAccesoPersonaLocal(aGuid : String, aFechaHora : Datetime) : boolean
1
1
1
<<Interface>>
ControlarAccesoPersona_DAO
+EliminarAccesoPersonaLocal(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosPersonas() : ArrayList
+EliminarAccesosPersonas() : void
+RegistrarAccesoPersona(ListaAcceso : ArrayList, Nombre_Cliente : String) : boolean
+RegistrarAccesoPersonaLocal(accesosPersona : AccesoPersona) : boolean
<<implementationClass>>
ControlarAccesoPersona_DAO_Imple
+EliminarAccesoPersonaLocal(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosPersonas() : ArrayList
+EliminarAccesosPersonas() : void
+RegistrarAccesoPersona(ListaAcceso : ArrayList, Nombre_Cliente : String) : boolean
+RegistrarAccesoPersonaLocal(accesosPersona : AccesoPersona) : boolean


<<implementationClass>>
CC_ControlarPersona_Imple
+BuscarPersona(Solapin : String, CodBarras : String, CI : String): Persona
+Auto_Registrar(CodBarra : String) : Persona
+Verificar_PassBack(aGuid : String, TipoAcceso : String) : boolean
+RegistrarPassback(passBack : PassBack) : void
+SalvarAcceso(persona : Persona, Tipo Acceso : String): void
+Deshacer(Guid : String, Ult_Fecha : Datetime): boolean
+BuscarCredenciales(aTipoCredencial : String): ArrayList
+BuscarTiposCredencial() : ArrayList
1
1
1
<<implementationClass>>
CC_Control_Configuracion_Imple
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime): DateTime
+BuscarPuertasNoActivas(): ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String): boolean
+ObtenerCampoLocal(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P_Sinc...
+ActualizarCampoLocal(aCampo : String): boolean
+ActualizarAccesosPersonas() : boolean
+ActualizarAccesosVehiculos() : boolean
+ActualizarAccesosVisitantes() : boolean
+ObtenerCausaAcceso() : ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean
1
1
<<implementationClass>>
CC_Control_Persona_Imple
+BuscarPersona(Solapin : String, CodBarras : String, CI : String): Persona
+BuscarCredenciales(TipoCredencial : String) : ArrayList
+BuscarTiposCredenciales() : ArrayList
1
1


1
1
<<Interface>>
Control_Configuracion_DAO
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime): DateTime
+BuscarPuertasNoActivas(): ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String): boolean
+ObtenerCampoLocal(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P_Sinc...
+ActualizarCampoLocal(aCampo : String): boolean
+ObtenerCausaAcceso(): ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean
1
<<Interface>>
Controlar_Persona_DAO
+BuscarPersona(aCodigoBarras : String, aNumeroUCI : String, aCI : String) : Persona
+BuscarCredenciales(aTipoCredencial : String): ArrayList
+BuscarTiposCredencial() : ArrayList
+EliminarPersonas() : boolean
+RegistrarPersonas(aListaPersonas : ArrayList): void
+BuscarPersonas() : ArrayList
<<implementationClass>>
Control_Configuracion_DAO_Imple
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime): DateTime
+BuscarPuertasNoActivas(): ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String): boolean
+ObtenerCampoLocal(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P_Sinc...
+ActualizarCampoLocal(aCampo : String): boolean
Controlar_Persona_DAO_Imple
+BuscarPersona(aCodigoBarras : String, aNumeroUCI : String, aCI : String) : Persona
+BuscarCredenciales(aTipoCredencial : String): ArrayList
+BuscarTiposCredencial() : ArrayList
+EliminarPersonas() : boolean
+RegistrarPersonas(aListaPersonas : ArrayList): void
+BuscarPersonas() : ArrayList

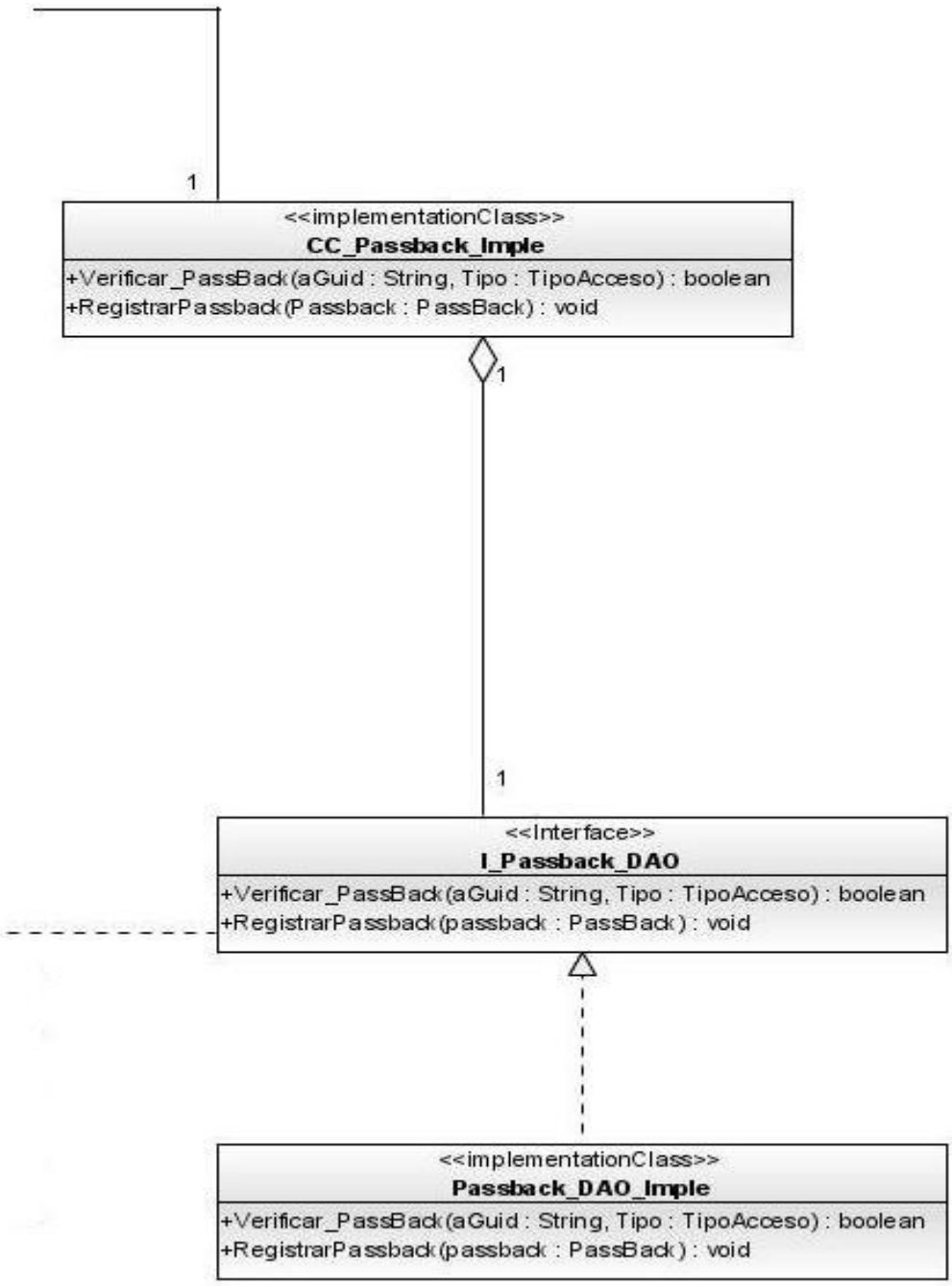
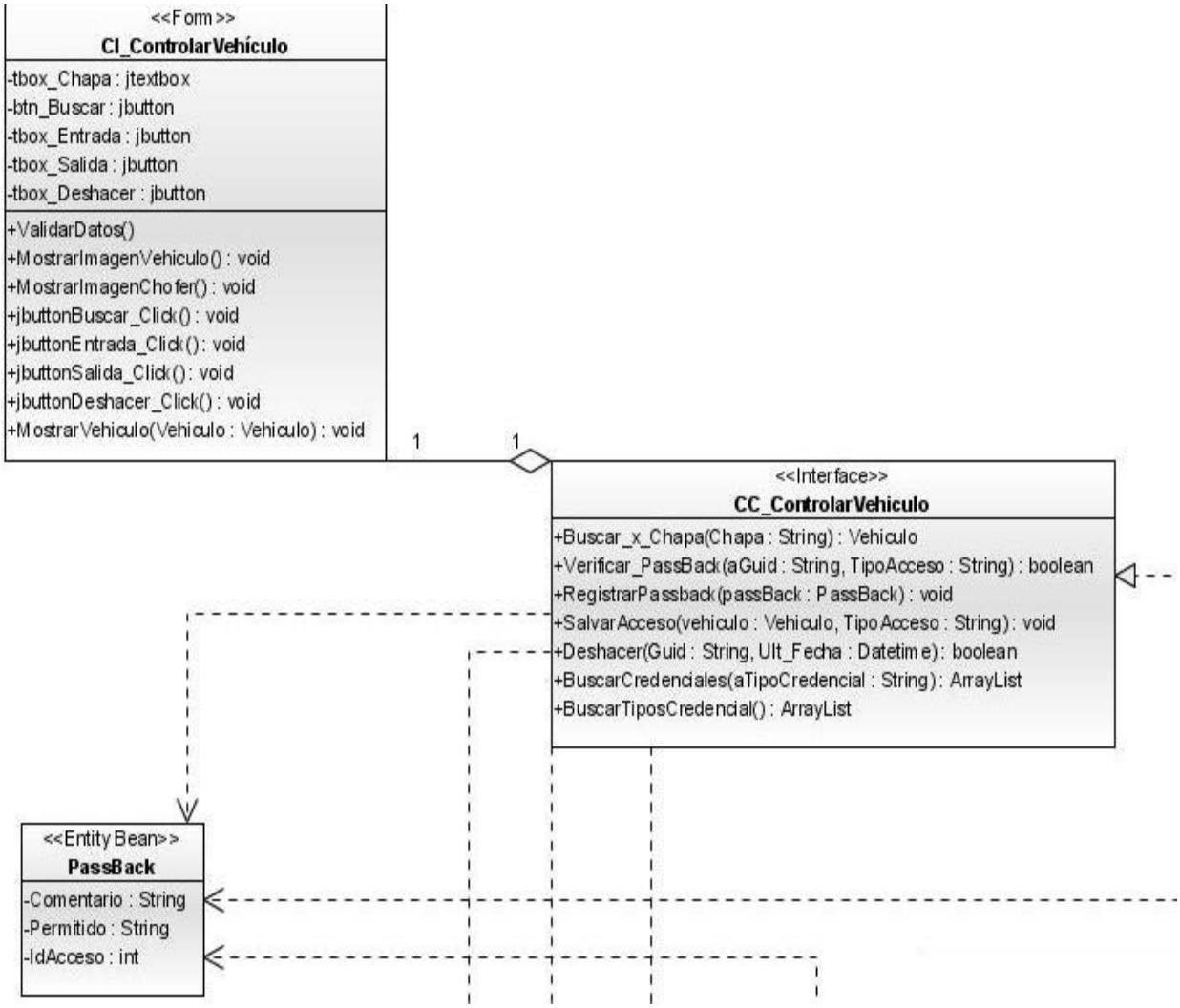
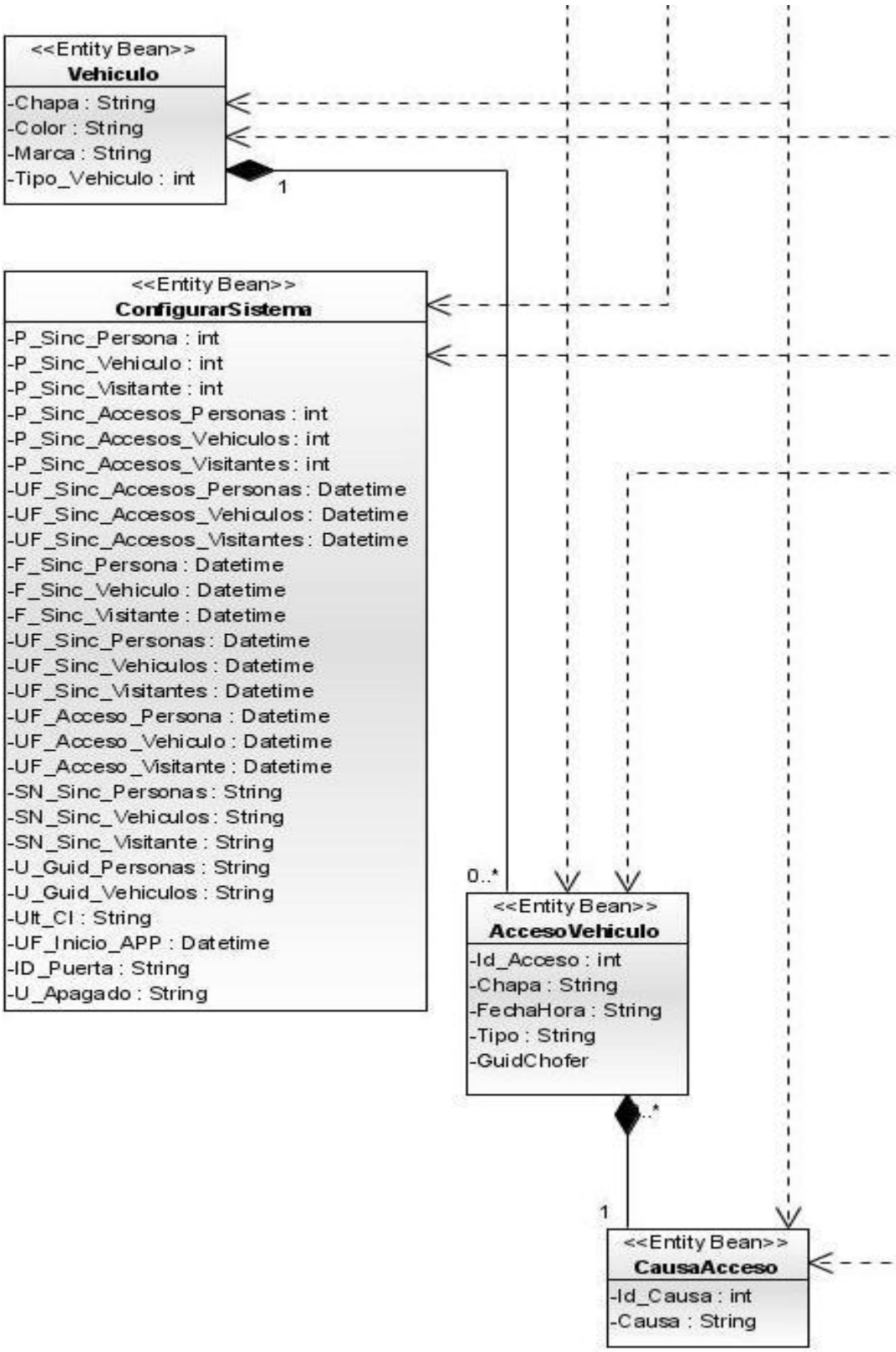
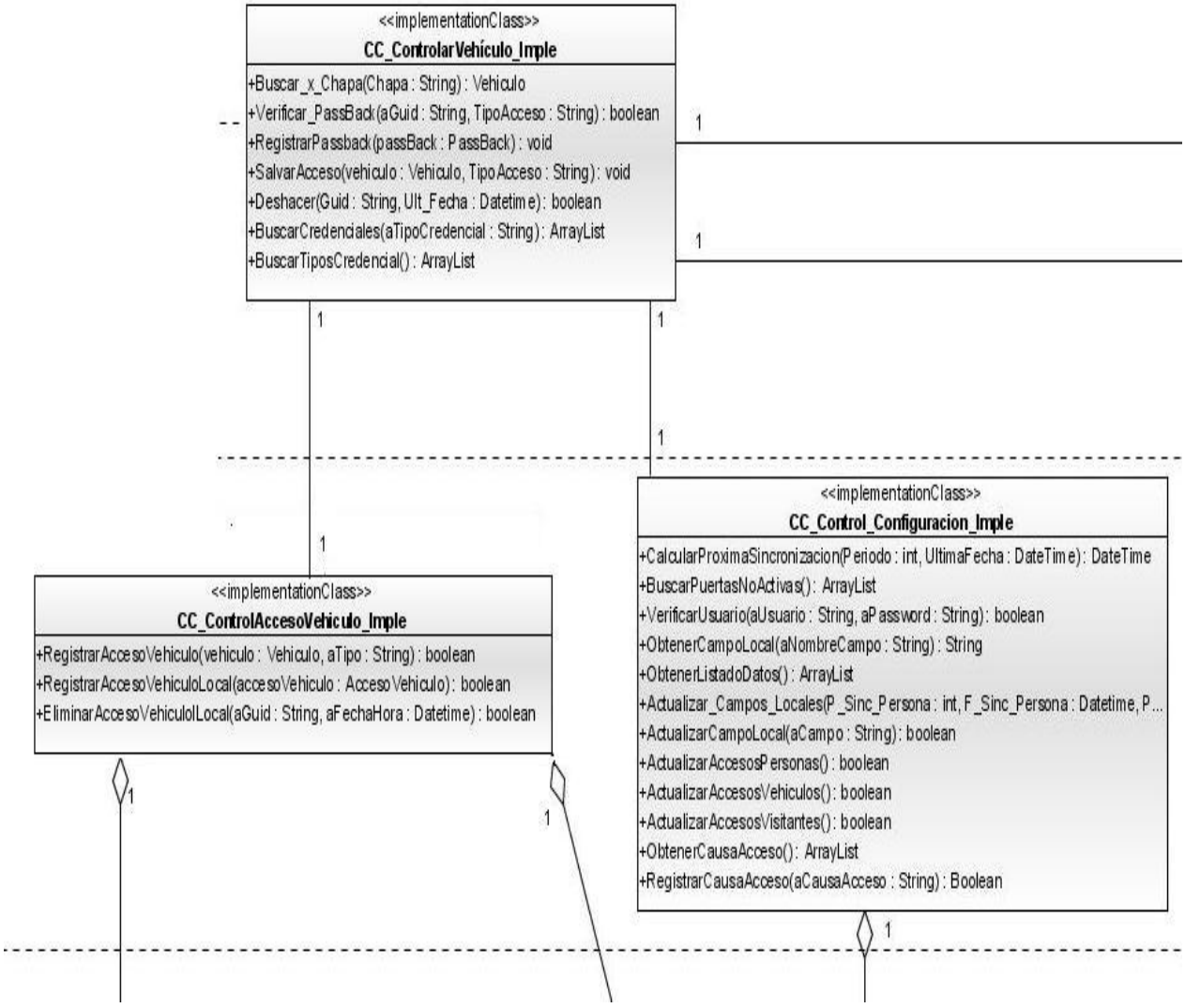
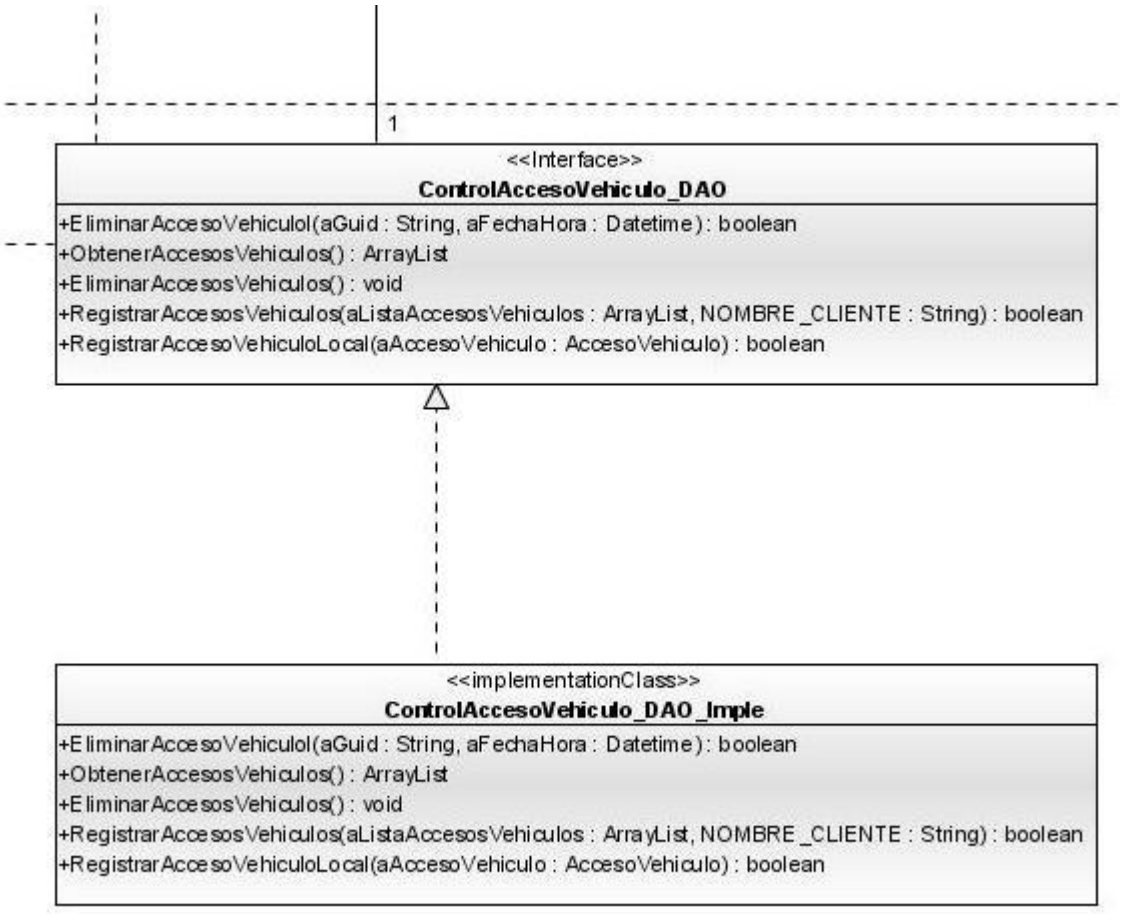


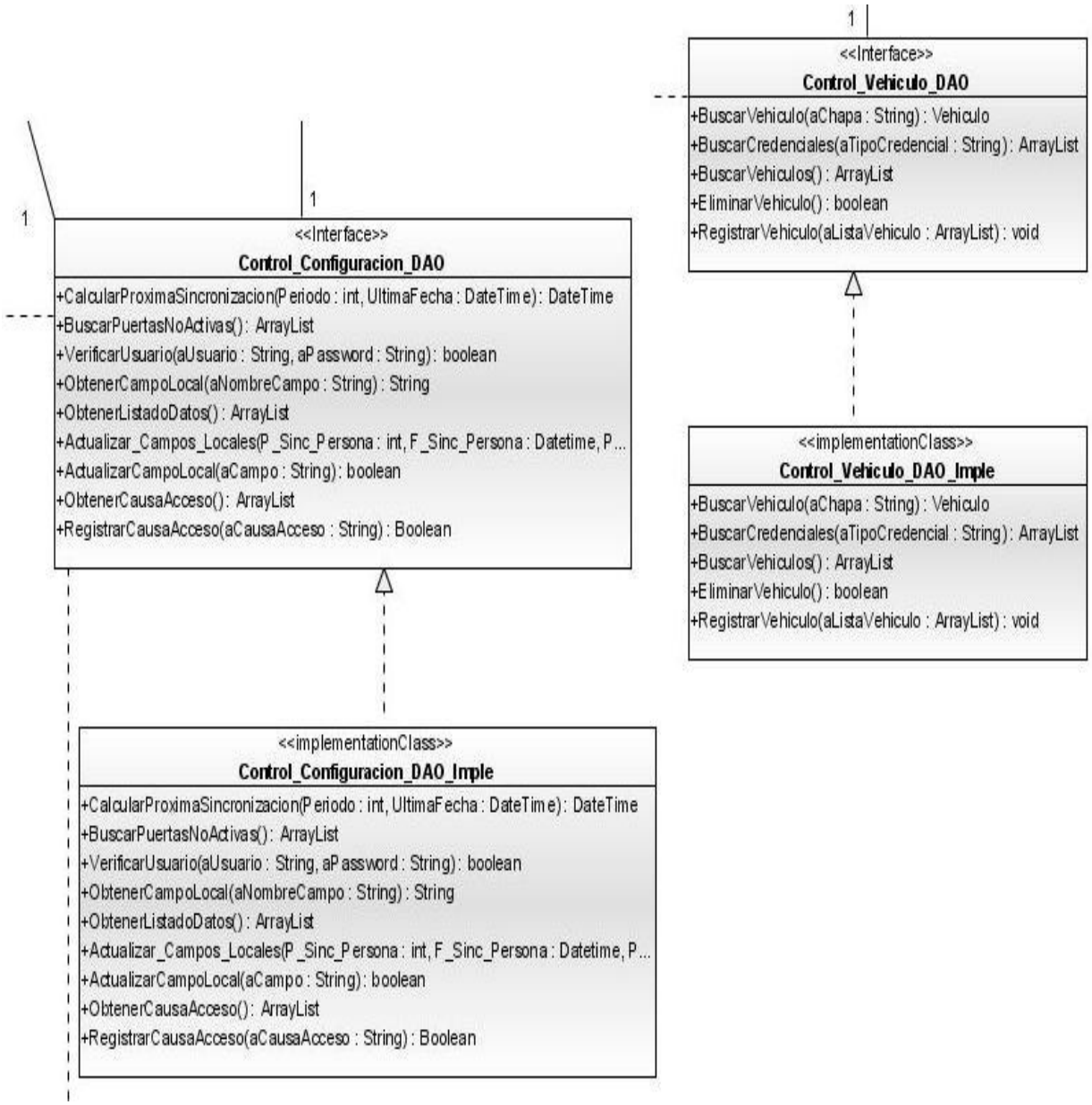
Figura 10 Diagrama de clase del diseño "Controlar Acceso puntual de Personas de la UCI"


<<Form>>
CI_ControlarVehículo
-tbox_Chapa : jtextbox
-btn_Buscar : jbutton
-tbox_Entrada : jbutton
-tbox_Salida : jbutton
-tbox_Deshacer : jbutton
+ValidarDatos()
+MostrarImagenVehiculo() : void
+MostrarImagenChofer() : void
+jbuttonBuscar_Click() : void
+jbuttonEntrada_Click() : void
+jbuttonSalida_Click() : void
+jbuttonDeshacer_Click() : void
+MostrarVehiculo(Vehiculo : Vehiculo) : void
1
1
<<Interface>>
CC_ControlarVehiculo
+Buscar_x_Chapa(Chapa : String) : Vehiculo
+Verificar_PassBack(aGuid : String, TipoAcceso : String) : boolean
+RegistrarPassback(passBack : PassBack) : void
+SalvarAcceso(vehiculo : Vehiculo, TipoAcceso : String) : void
+Deshacer(Guid : String, Ult_Fecha : Datetime) : boolean
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarTiposCredencial() : ArrayList
<<Entity Bean>>
PassBack
-Comentario : String
-Permitido : String
-IdAcceso : int


<<Entity Bean>>
Vehiculo
-Chapa : String
-Color : String
-Marca : String
-Tipo_Vehiculo : int
1
<<Entity Bean>>
ConfigurarSistema
-P_Sinc_Persona : int
-P_Sinc_Vehiculo : int
-P_Sinc_Visitante : int
-P_Sinc_Accesos_Personas : int
-P_Sinc_Accesos_Vehiculos : int
-P_Sinc_Accesos_Visitantes : int
-UF_Sinc_Accesos_Personas : Datetime
-UF_Sinc_Accesos_Vehiculos : Datetime
-UF_Sinc_Accesos_Visitantes : Datetime
-F_Sinc_Persona : Datetime
-F_Sinc_Vehiculo : Datetime
-F_Sinc_Visitante : Datetime
-UF_Sinc_Personas : Datetime
-UF_Sinc_Vehiculos : Datetime
-UF_Sinc_Visitantes : Datetime
-UF_Acceso_Persona : Datetime
-UF_Acceso_Vehiculo : Datetime
-UF_Acceso_Visitante : Datetime
-SN_Sinc_Personas : String
-SN_Sinc_Vehiculos : String
-SN_Sinc_Visitante : String
-U_Guid_Personas : String
-U_Guid_Vehiculos : String
-Ult_CI : String
-UF_Inicio_APP : Datetime
-ID_Puerta : String
-U_Apagado : String
0..*
<<Entity Bean>>
AccesoVehiculo
-Id_Acceso : int
-Chapa : String
-FechaHora : String
-Tipo : String
-GuidChofer
..*
1
<<Entity Bean>>
CausaAcceso
-Id_Causa : int
-Causa : String


<<implementationClass>>
CC_ControlarVehículo_Imple
+Buscar_x_Chapa(Chapa : String) : Vehiculo
+Verificar_PassBack(aGuid : String, TipoAcceso : String) : boolean
+RegistrarPassback(passBack : PassBack) : void
+SalvarAcceso(vehiculo : Vehiculo, TipoAcceso : String) : void
+Deshacer(Guid : String, Ult_Fecha : Datetime) : boolean
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarTiposCredencial() : ArrayList
1
1
1
1
1
1
<<implementationClass>>
CC_ControlAccesoVehiculo_Imple
+RegistrarAccesoVehiculo(vehiculo : Vehiculo, aTipo : String) : boolean
+RegistrarAccesoVehiculoLocal(accesoVehiculo : AccesoVehiculo) : boolean
+EliminarAccesoVehiculolLocal(aGuid : String, aFechaHora : Datetime) : boolean
1
1
<<implementationClass>>
CC_Control_Configuracion_Imple
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime) : DateTime
+BuscarPuertasNoActivas() : ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String) : boolean
+ObtenerCampoLocal(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P...
+ActualizarCampoLocal(aCampo : String) : boolean
+ActualizarAccesosPersonas() : boolean
+ActualizarAccesosVehiculos() : boolean
+ActualizarAccesosVisitantes() : boolean
+ObtenerCausaAcceso() : ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean
1


1
<<Interface>>
ControlAccesoVehiculo_DAO
+EliminarAccesoVehiculoI(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosVehiculos() : ArrayList
+EliminarAccesosVehiculos() : void
+RegistrarAccesosVehiculos(aListaAccesosVehiculos : ArrayList, NOMBRE_CLIENTE : String) : boolean
+RegistrarAccesoVehiculoLocal(aAccesoVehiculo : AccesoVehiculo) : boolean
<<implementationClass>>
ControlAccesoVehiculo_DAO_Imple
+EliminarAccesoVehiculoI(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosVehiculos() : ArrayList
+EliminarAccesosVehiculos() : void
+RegistrarAccesosVehiculos(aListaAccesosVehiculos : ArrayList, NOMBRE_CLIENTE : String) : boolean
+RegistrarAccesoVehiculoLocal(aAccesoVehiculo : AccesoVehiculo) : boolean


1
<<Interface>>
Control_Vehiculo_DAO
+BuscarVehiculo(aChapa : String) : Vehiculo
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarVehiculos() : ArrayList
+EliminarVehiculo() : boolean
+RegistrarVehiculo(aListaVehiculo : ArrayList) : void
1
1
<<Interface>>
Control_Configuracion_DAO
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime) : DateTime
+BuscarPuertasNoActivas() : ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String) : boolean
+ObtenerCampoLocal(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P...
+ActualizarCampoLocal(aCampo : String) : boolean
+ObtenerCausaAcceso() : ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean
<<implementationClass>>
Control_Vehiculo_DAO_Imple
+BuscarVehiculo(aChapa : String) : Vehiculo
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarVehiculos() : ArrayList
+EliminarVehiculo() : boolean
+RegistrarVehiculo(aListaVehiculo : ArrayList) : void
<<implementationClass>>
Control_Configuracion_DAO_Imple
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime) : DateTime
+BuscarPuertasNoActivas() : ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String) : boolean
+ObtenerCampoLocal(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P...
+ActualizarCampoLocal(aCampo : String) : boolean
+ObtenerCausaAcceso() : ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean

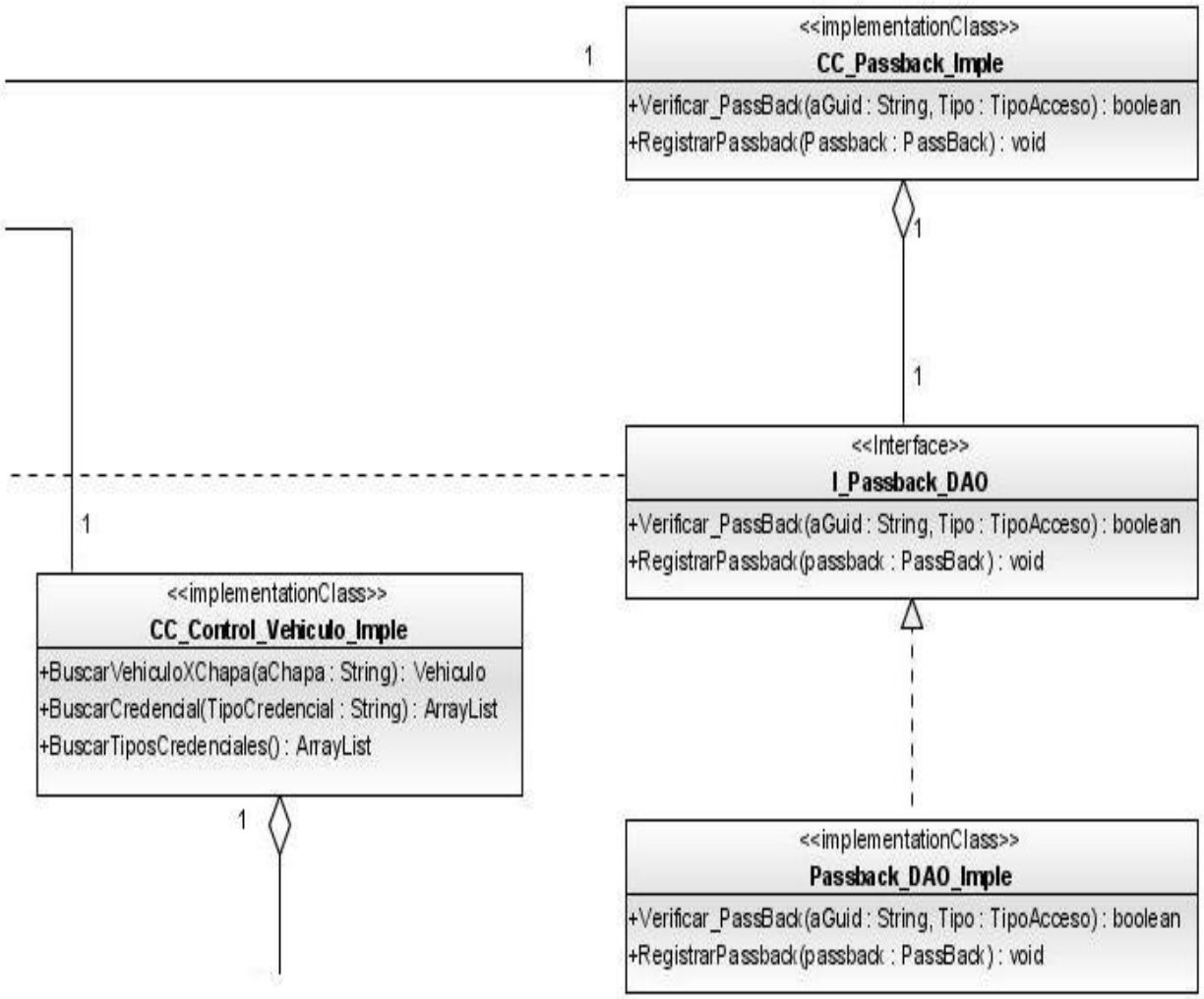
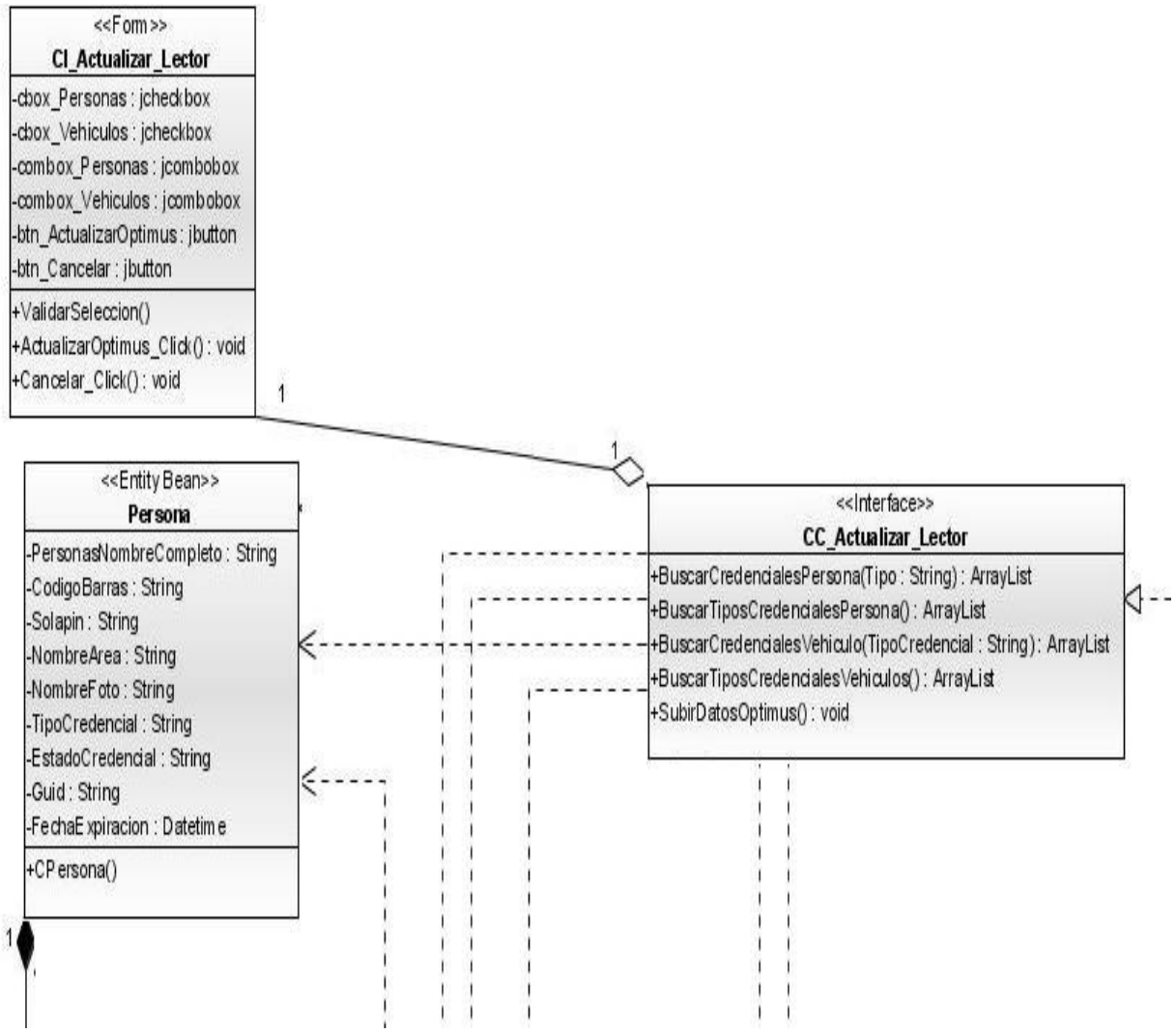
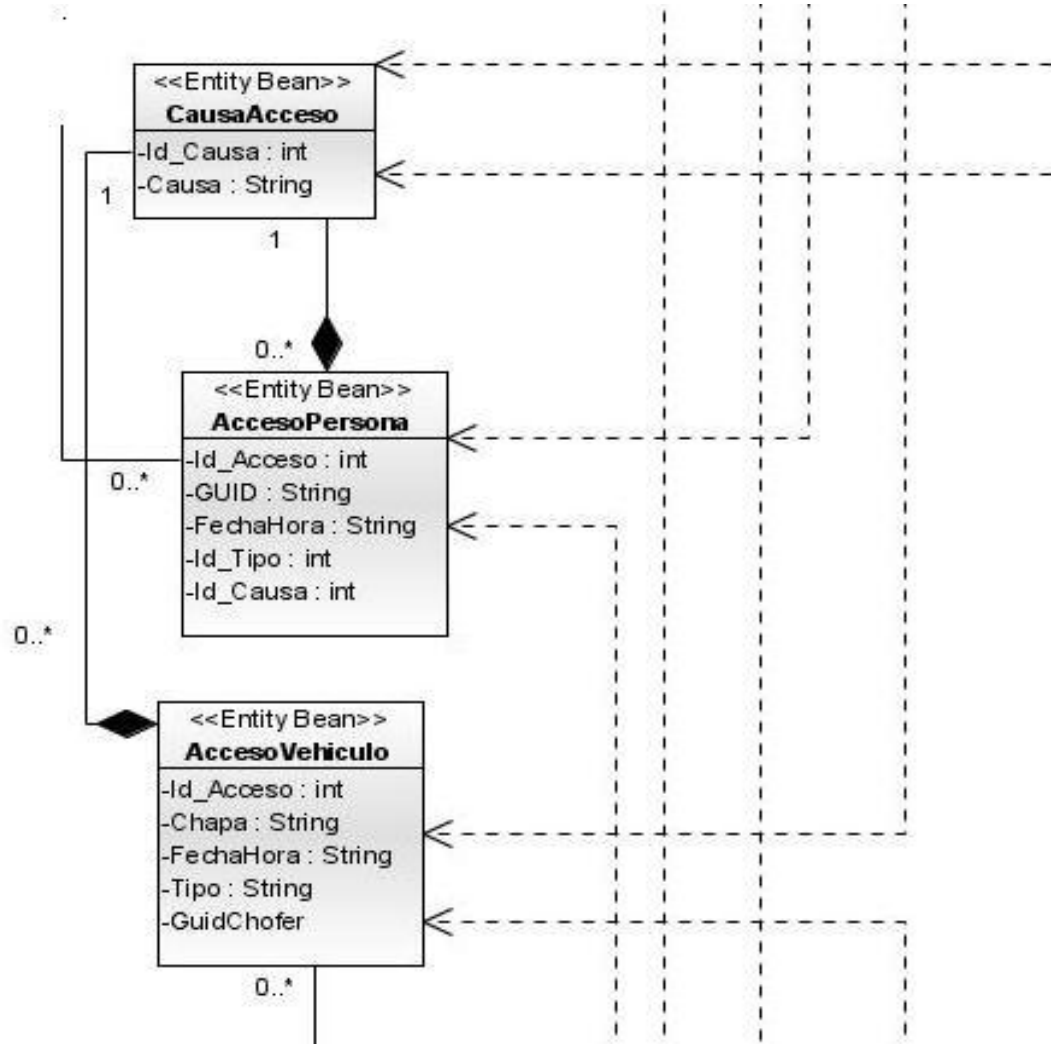
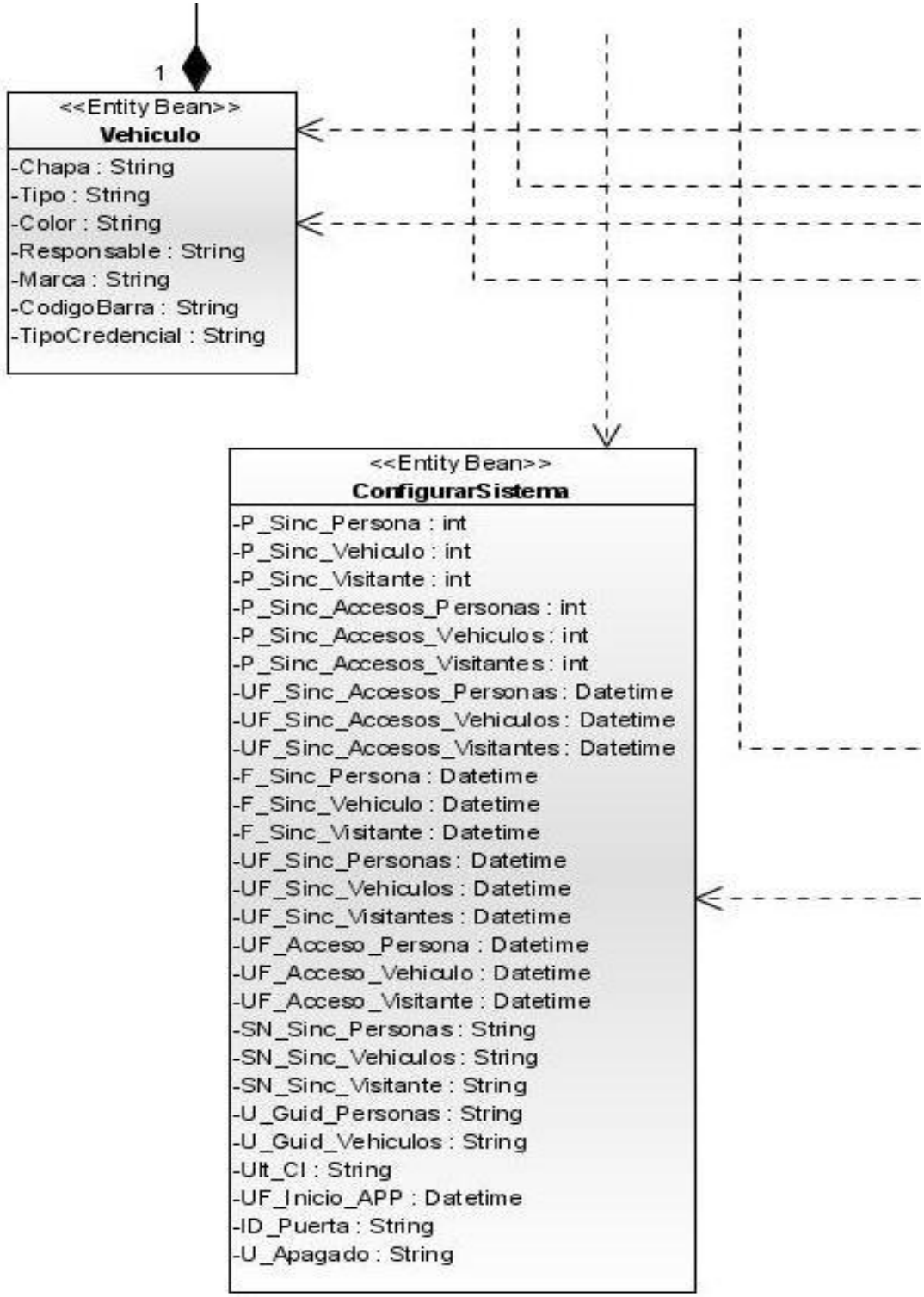
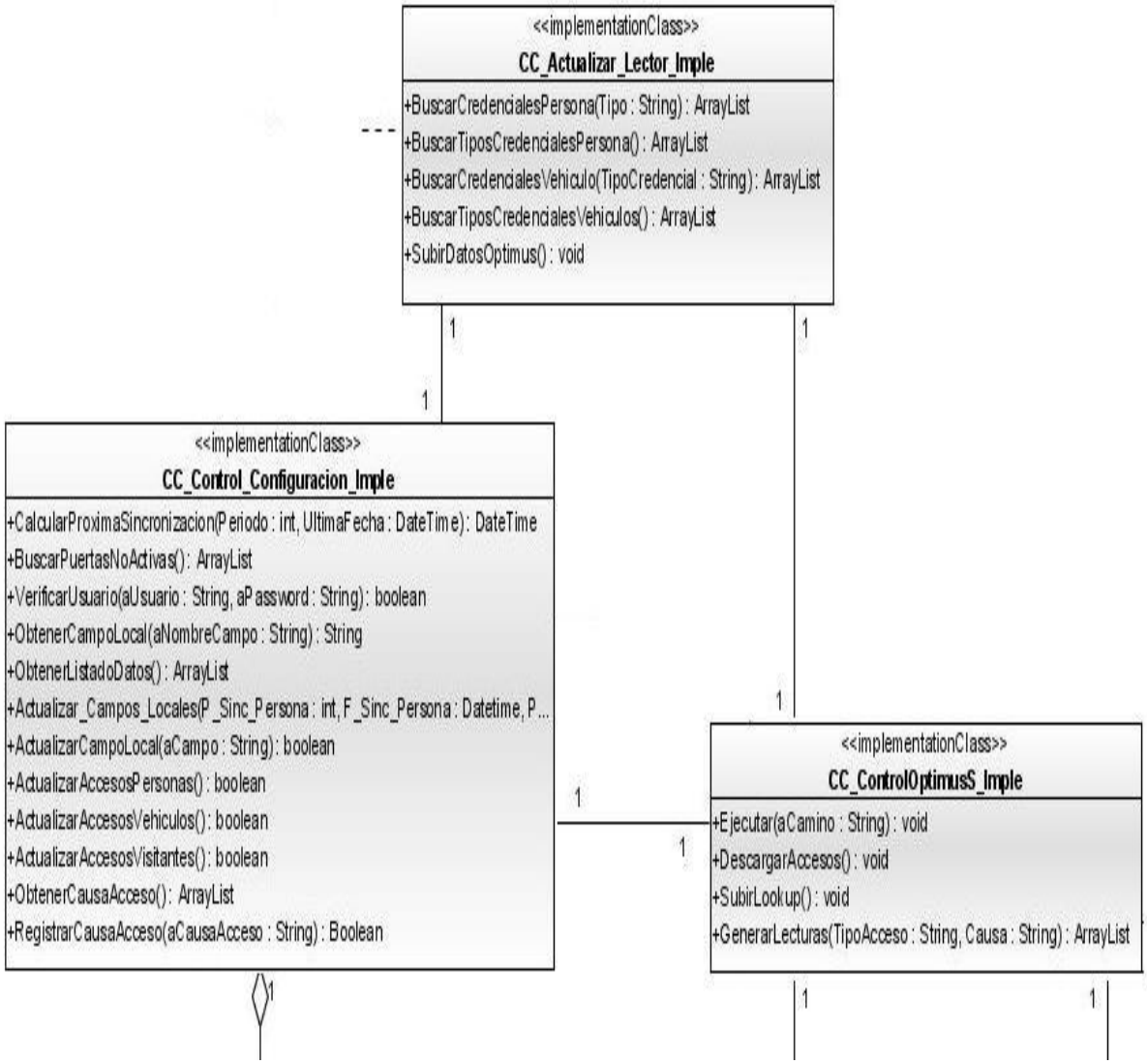


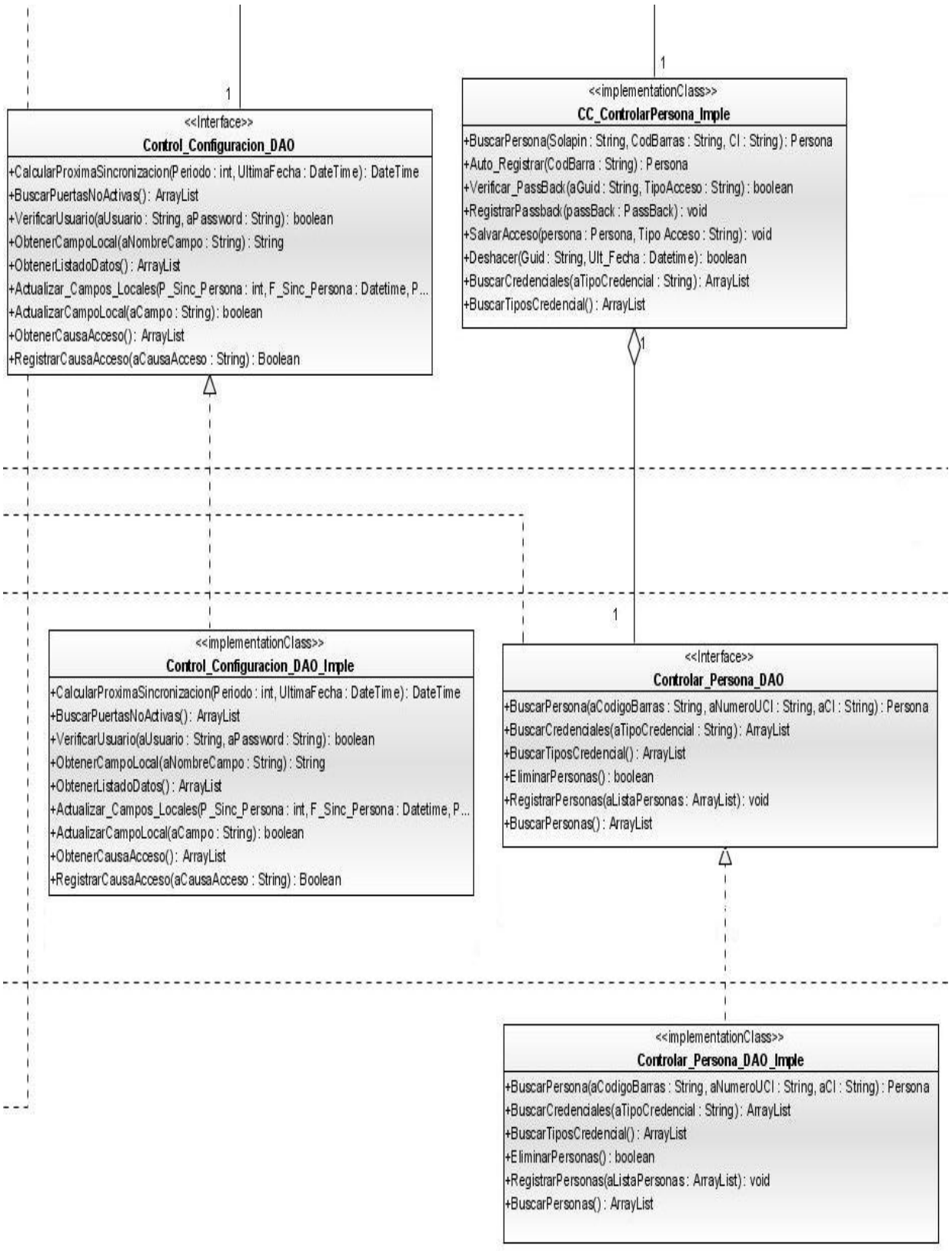
Figura 11 Diagrama de clase del diseño "Controlar Acceso de Vehículo"

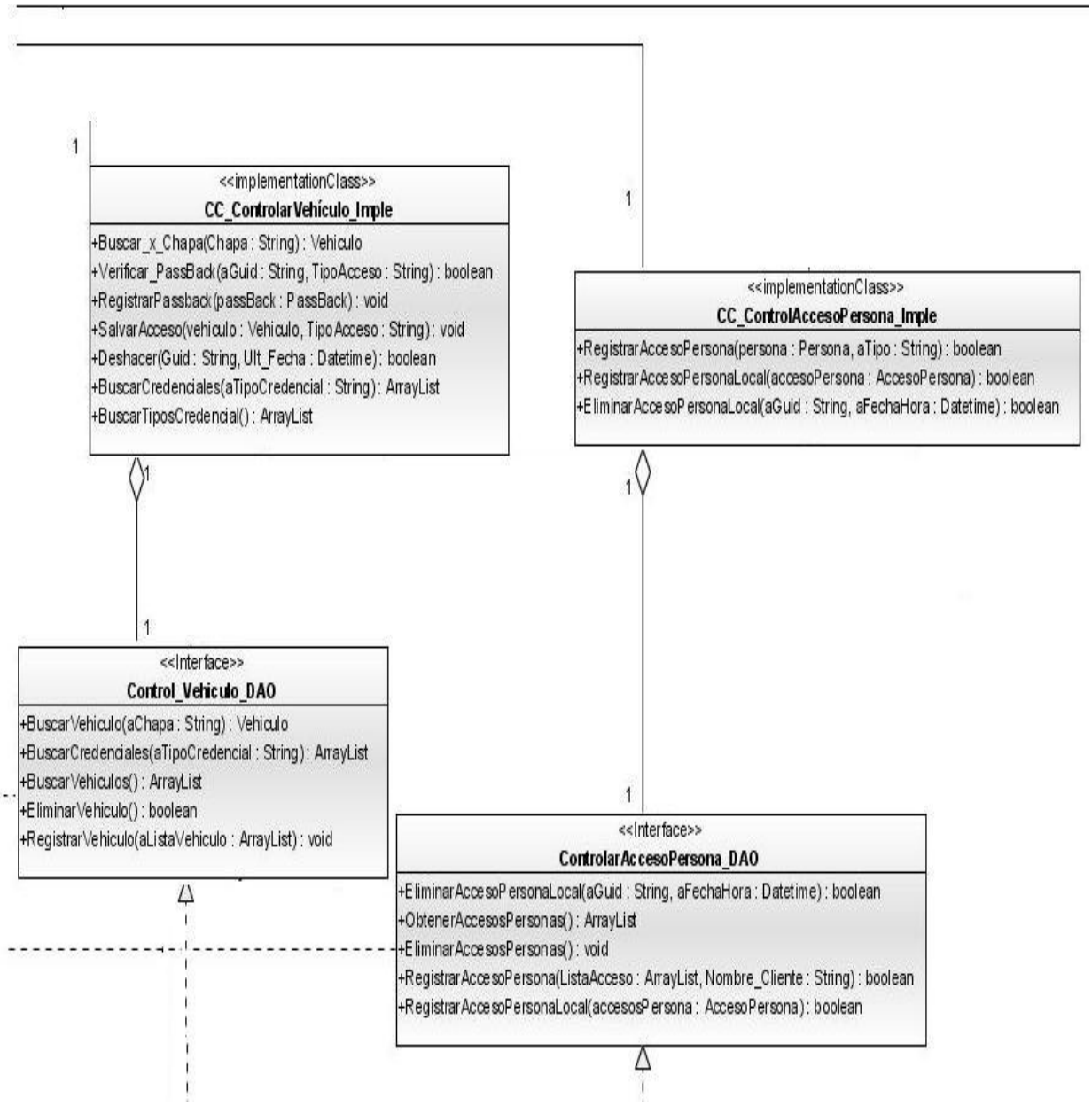

<<Form>>
CI_Actualizar_Lector
-cbox_Personas : jcheckbox
-cbox_Vehiculos : jcheckbox
-combox_Personas : jcombobox
-combox_Vehiculos : jcombobox
-btn_ActualizarOptimus : jbutton
-btn_Cancelar : jbutton
+ValidarSeleccion()
+ActualizarOptimus_Click() : void
+Cancelar_Click() : void
1
1
<<Entity Bean>>
Persona
-PersonasNombreCompleto : String
-CodigoBarras : String
-Solapin : String
-NombreArea : String
-NombreFoto : String
-TipoCredencial : String
-EstadoCredencial : String
-Guid : String
-FechaExpiracion : Datetime
+CPersona()
1
<<Interface>>
CC_Actualizar_Lector
+BuscarCredencialesPersona(Tipo : String) : ArrayList
+BuscarTiposCredencialesPersona() : ArrayList
+BuscarCredencialesVehiculo(TipoCredencial : String) : ArrayList
+BuscarTiposCredencialesVehiculos() : ArrayList
+SubirDatosOptimus() : void

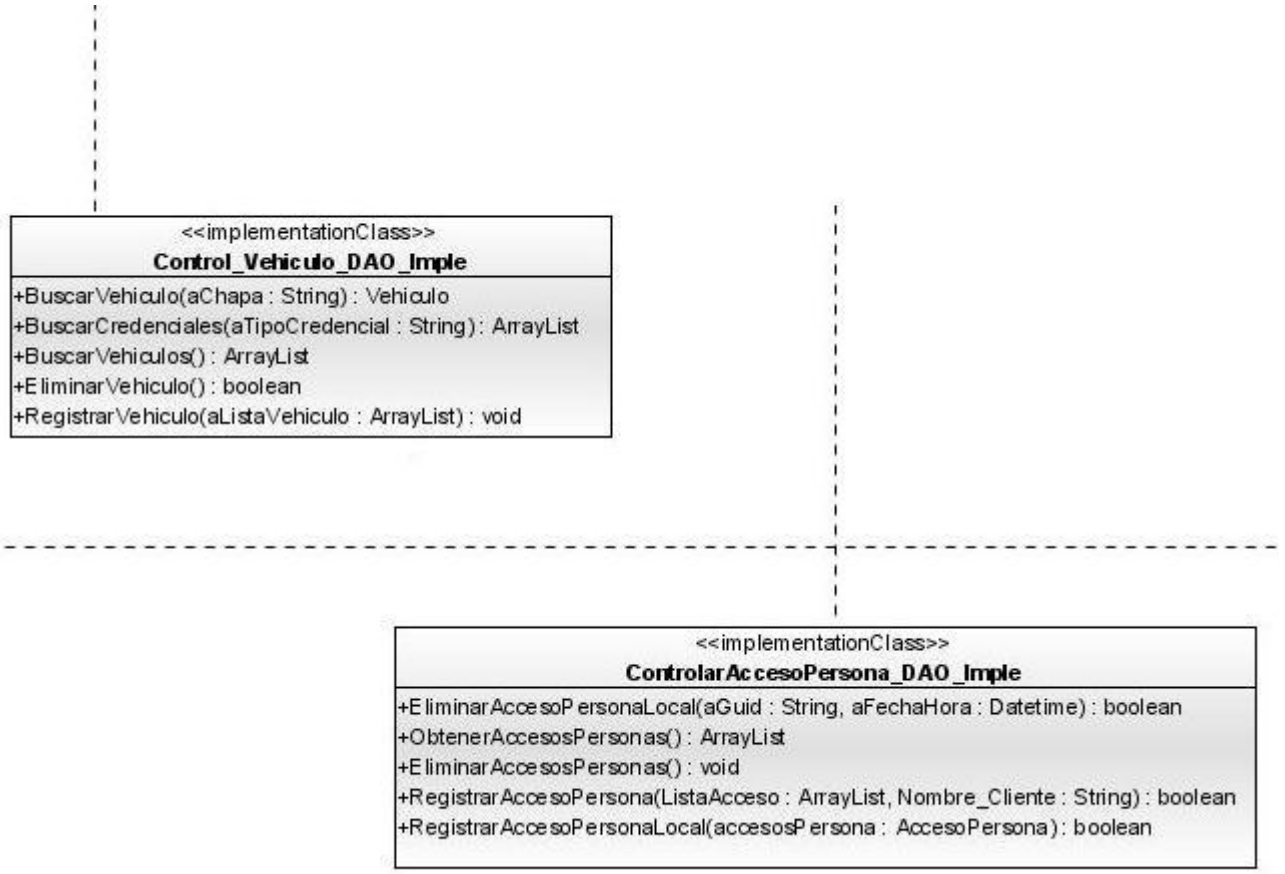

<<Entity Bean>>
CausaAcceso
-Id_Causa : int
-Causa : String
1
1
0..*
<<Entity Bean>>
AccesoPersona
-Id_Acceso : int
-GUID : String
-FechaHora : String
-Id_Tipo : int
-Id_Causa : int
0..*
0..*
<<Entity Bean>>
AccesoVehiculo
-Id_Acceso : int
-Chapa : String
-FechaHora : String
-Tipo : String
-GuidChofer
0..*


1
<<Entity Bean>>
Vehiculo
-Chapa : String
-Tipo : String
-Color : String
-Responsable : String
-Marca : String
-CodigoBarra : String
-TipoCredencial : String
<<Entity Bean>>
ConfigurarSistema
-P_Sinc_Persona : int
-P_Sinc_Vehiculo : int
-P_Sinc_Visitante : int
-P_Sinc_Accesos_Personas : int
-P_Sinc_Accesos_Vehiculos : int
-P_Sinc_Accesos_Visitantes : int
-UF_Sinc_Accesos_Personas : Datetime
-UF_Sinc_Accesos_Vehiculos : Datetime
-UF_Sinc_Accesos_Visitantes : Datetime
-F_Sinc_Persona : Datetime
-F_Sinc_Vehiculo : Datetime
-F_Sinc_Visitante : Datetime
-UF_Sinc_Personas : Datetime
-UF_Sinc_Vehiculos : Datetime
-UF_Sinc_Visitantes : Datetime
-UF_Acceso_Persona : Datetime
-UF_Acceso_Vehiculo : Datetime
-UF_Acceso_Visitante : Datetime
-SN_Sinc_Personas : String
-SN_Sinc_Vehiculos : String
-SN_Sinc_Visitante : String
-U_Guid_Personas : String
-U_Guid_Vehiculos : String
-Ult_CI : String
-UF_Inicio_APP : Datetime
-ID_Puerta : String
-U_Apagado : String


<<implementationClass>>
CC_Actualizar_Lector_Imple
+BuscarCredencialesPersona(Tipo : String) : ArrayList
+BuscarTiposCredencialesPersona() : ArrayList
+BuscarCredencialesVehiculo(TipoCredencial : String) : ArrayList
+BuscarTiposCredencialesVehiculos() : ArrayList
+SubirDatosOptimus() : void
<<implementationClass>>
CC_Control_Configuracion_Imple
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime) : DateTime
+BuscarPuertasNoActivas() : ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String) : boolean
+ObtenerCampoLocal(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P...
+ActualizarCampoLocal(aCampo : String) : boolean
+ActualizarAccesosPersonas() : boolean
+ActualizarAccesosVehiculos() : boolean
+ActualizarAccesosVisitantes() : boolean
+ObtenerCausaAcceso() : ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean
<<implementationClass>>
CC_ControlOptimusS_Imple
+Ejecutar(aCamino : String) : void
+DescargarAccesos() : void
+SubirLookup() : void
+GenerarLecturas(TipoAcceso : String, Causa : String) : ArrayList
1
1
1
1
1
1
1
1
1
1
1
1


1
<<Interface>>
Control_Configuracion_DAO
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime): DateTime
+BuscarPuertasNoActivas(): ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String): boolean
+ObtenerCampoLocal(aNombreCampo : String): String
+ObtenerListadoDatos(): ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P...
+ActualizarCampoLocal(aCampo : String): boolean
+ObtenerCausaAcceso(): ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean
1
<<implementationClass>>
CC_ControlarPersona_Imple
+BuscarPersona(Solapin : String, CodBarras : String, CI : String): Persona
+Auto_Registrar(CodBarra : String) : Persona
+Verificar_PassBack(aGuid : String, TipoAcceso : String) : boolean
+RegistrarPassback(passBack : PassBack) : void
+SalvarAcceso(persona : Persona, Tipo Acceso : String): void
+Deshacer(Guid : String, Ult_Fecha : Datetime): boolean
+BuscarCredenciales(aTipoCredencial : String): ArrayList
+BuscarTiposCredencial() : ArrayList
1
<<implementationClass>>
Control_Configuracion_DAO_Imple
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime): DateTime
+BuscarPuertasNoActivas(): ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String): boolean
+ObtenerCampoLocal(aNombreCampo : String): String
+ObtenerListadoDatos(): ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : Datetime, P...
+ActualizarCampoLocal(aCampo : String): boolean
+ObtenerCausaAcceso(): ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean
1
<<Interface>>
Controlar_Persona_DAO
+BuscarPersona(aCodigoBarras : String, aNumeroUCI : String, aCI : String) : Persona
+BuscarCredenciales(aTipoCredencial : String): ArrayList
+BuscarTiposCredencial() : ArrayList
+EliminarPersonas() : boolean
+RegistrarPersonas(aListaPersonas : ArrayList): void
+BuscarPersonas() : ArrayList
<<implementationClass>>
Controlar_Persona_DAO_Imple
+BuscarPersona(aCodigoBarras : String, aNumeroUCI : String, aCI : String) : Persona
+BuscarCredenciales(aTipoCredencial : String): ArrayList
+BuscarTiposCredencial() : ArrayList
+EliminarPersonas() : boolean
+RegistrarPersonas(aListaPersonas : ArrayList): void
+BuscarPersonas() : ArrayList


1
<<implementationClass>>
CC_ControlarVehículo_Imple
+Buscar_x_Chapa(Chapa : String) : Vehiculo
+Verificar_PassBack(aGuid : String, TipoAcceso : String) : boolean
+RegistrarPassback(passBack : PassBack) : void
+SalvarAcceso(vehiculo : Vehiculo, TipoAcceso : String) : void
+Deshacer(Guid : String, Ult_Fecha : Datetime) : boolean
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarTiposCredencial() : ArrayList
1
1
<<implementationClass>>
CC_ControlAccesoPersona_Imple
+RegistrarAccesoPersona(persona : Persona, aTipo : String) : boolean
+RegistrarAccesoPersonaLocal(accesoPersona : AccesoPersona) : boolean
+EliminarAccesoPersonaLocal(aGuid : String, aFechaHora : Datetime) : boolean
1
1
1
<<Interface>>
Control_Vehiculo_DAO
+BuscarVehiculo(aChapa : String) : Vehiculo
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarVehiculos() : ArrayList
+EliminarVehiculo() : boolean
+RegistrarVehiculo(aListaVehiculo : ArrayList) : void
1
<<Interface>>
ControlarAccesoPersona_DAO
+EliminarAccesoPersonaLocal(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosPersonas() : ArrayList
+EliminarAccesosPersonas() : void
+RegistrarAccesoPersona(ListaAcceso : ArrayList, Nombre_Cliente : String) : boolean
+RegistrarAccesoPersonaLocal(accesosPersona : AccesoPersona) : boolean


<<implementationClass>>
Control_Vehiculo_DAO_Imple
+BuscarVehiculo(aChapa : String) : Vehiculo
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarVehiculos() : ArrayList
+EliminarVehiculo() : boolean
+RegistrarVehiculo(aListaVehiculo : ArrayList) : void
<<implementationClass>>
ControlarAccesoPersona_DAO_Imple
+EliminarAccesoPersonaLocal(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosPersonas() : ArrayList
+EliminarAccesosPersonas() : void
+RegistrarAccesoPersona(ListaAcceso : ArrayList, Nombre_Cliente : String) : boolean
+RegistrarAccesoPersonaLocal(accesosPersona : AccesoPersona) : boolean

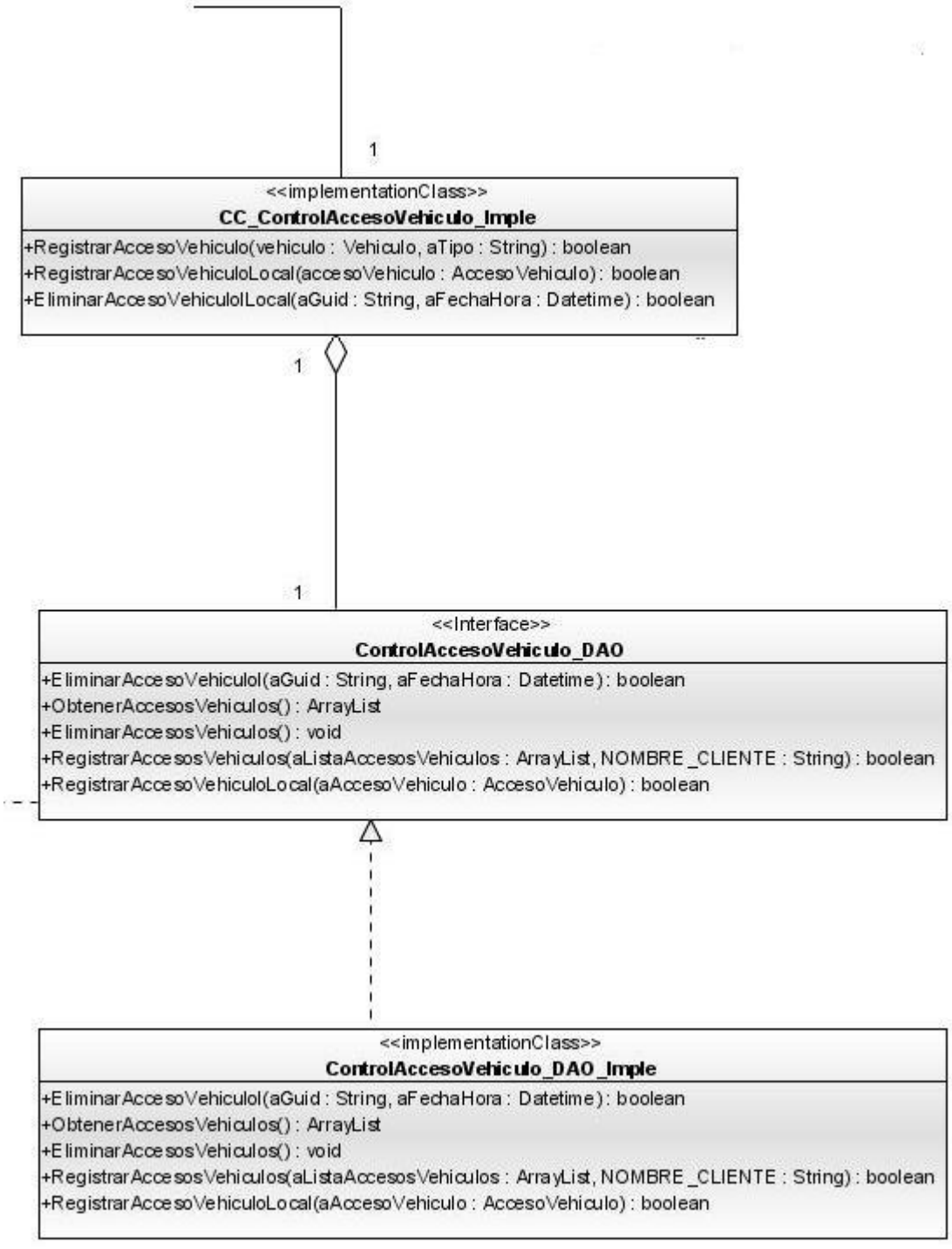
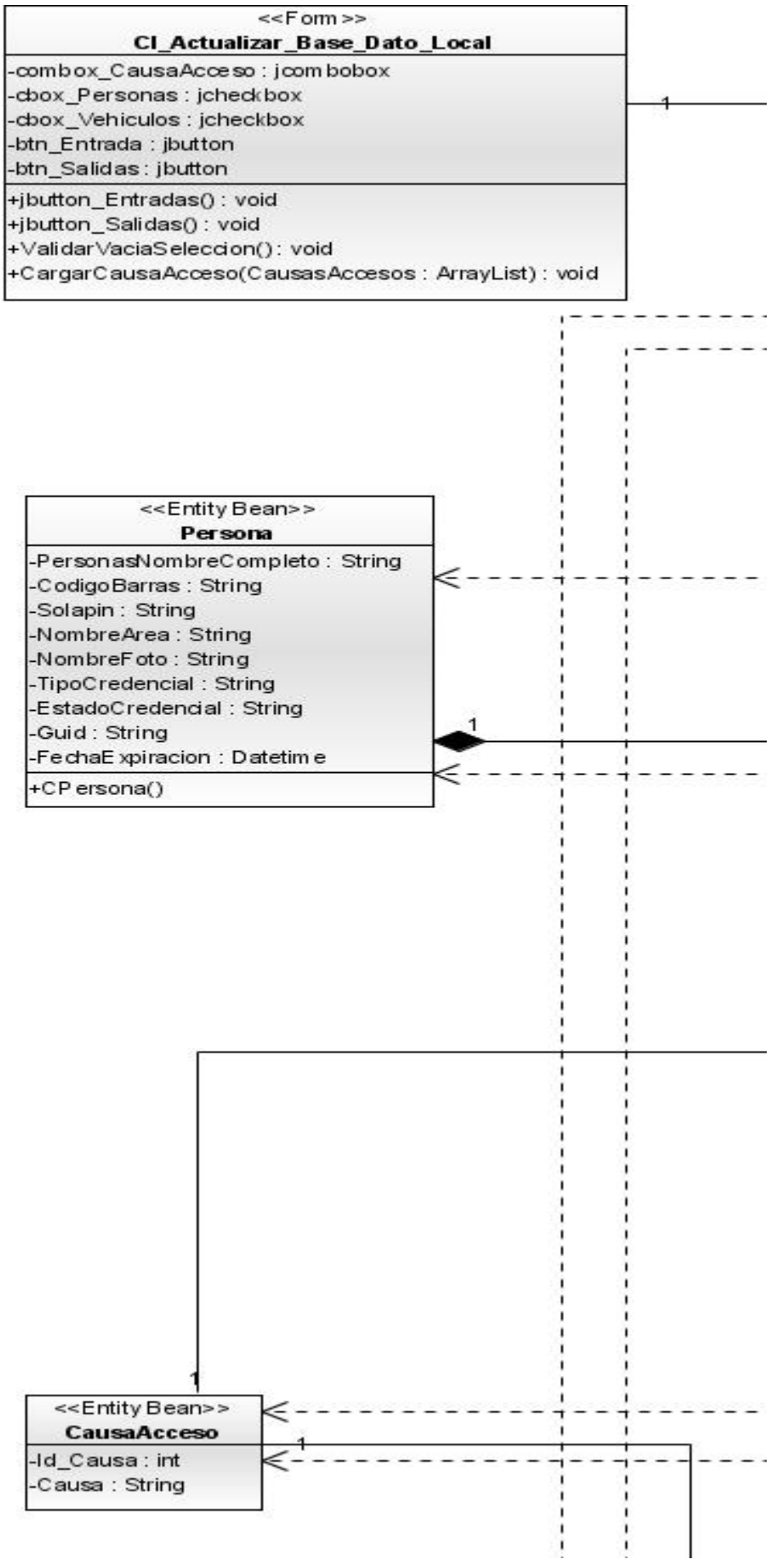
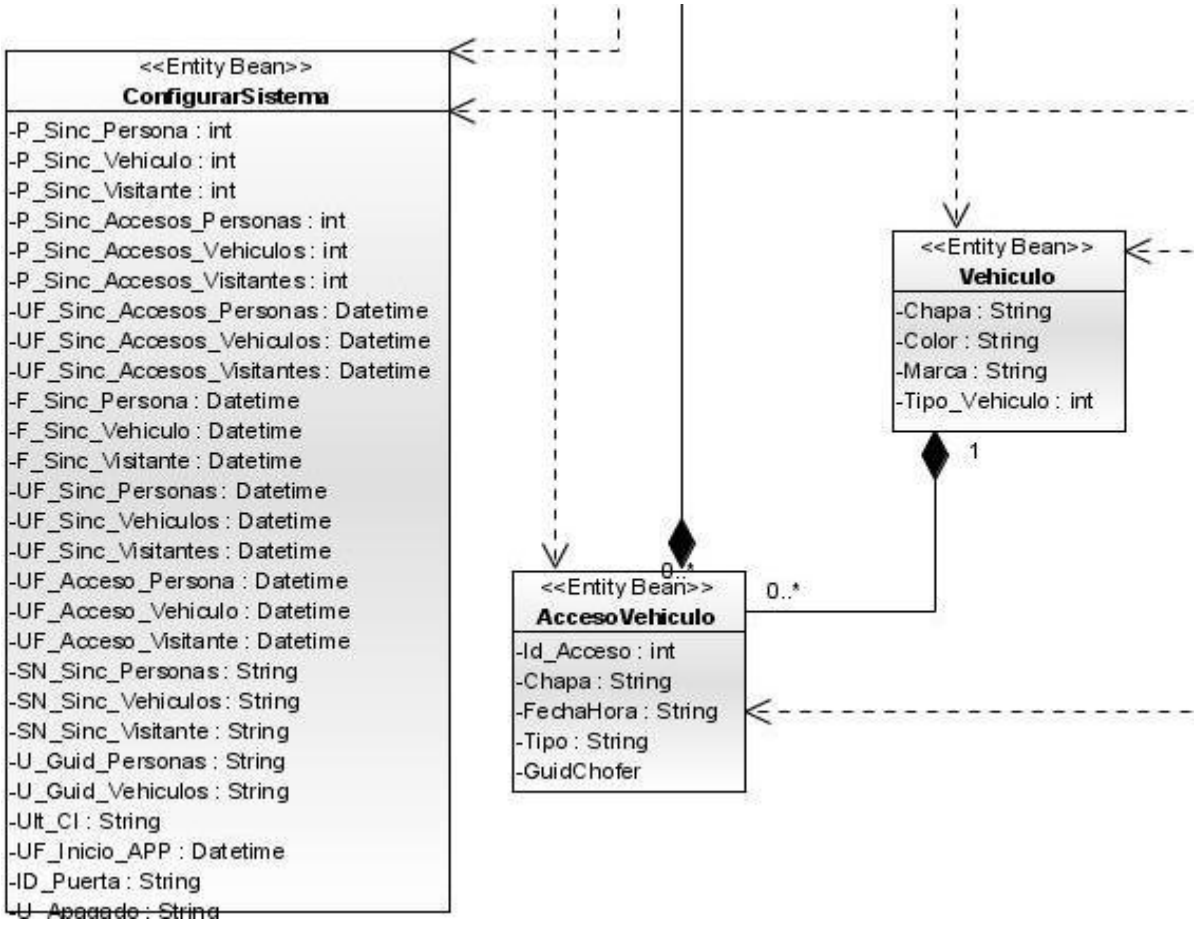
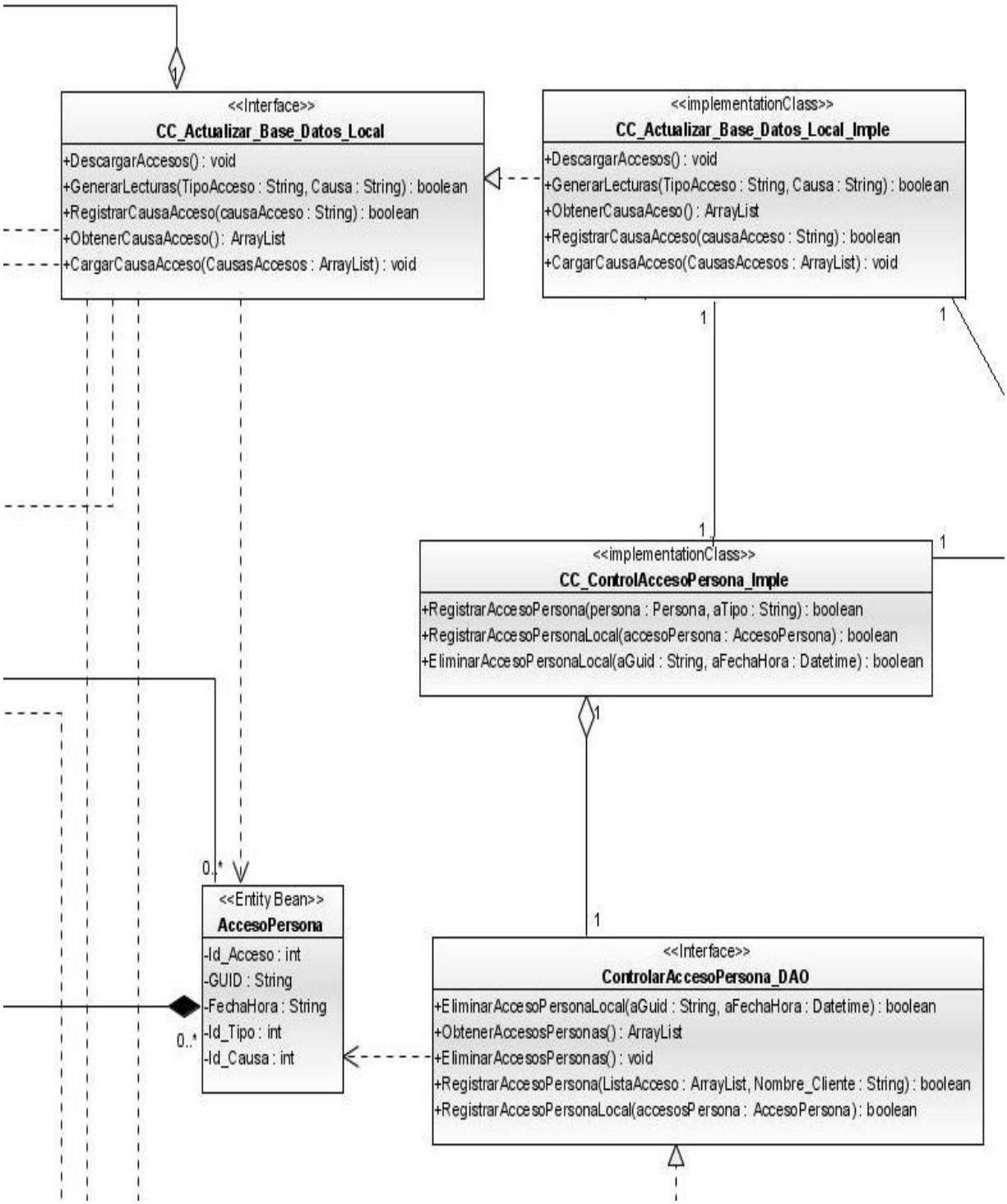
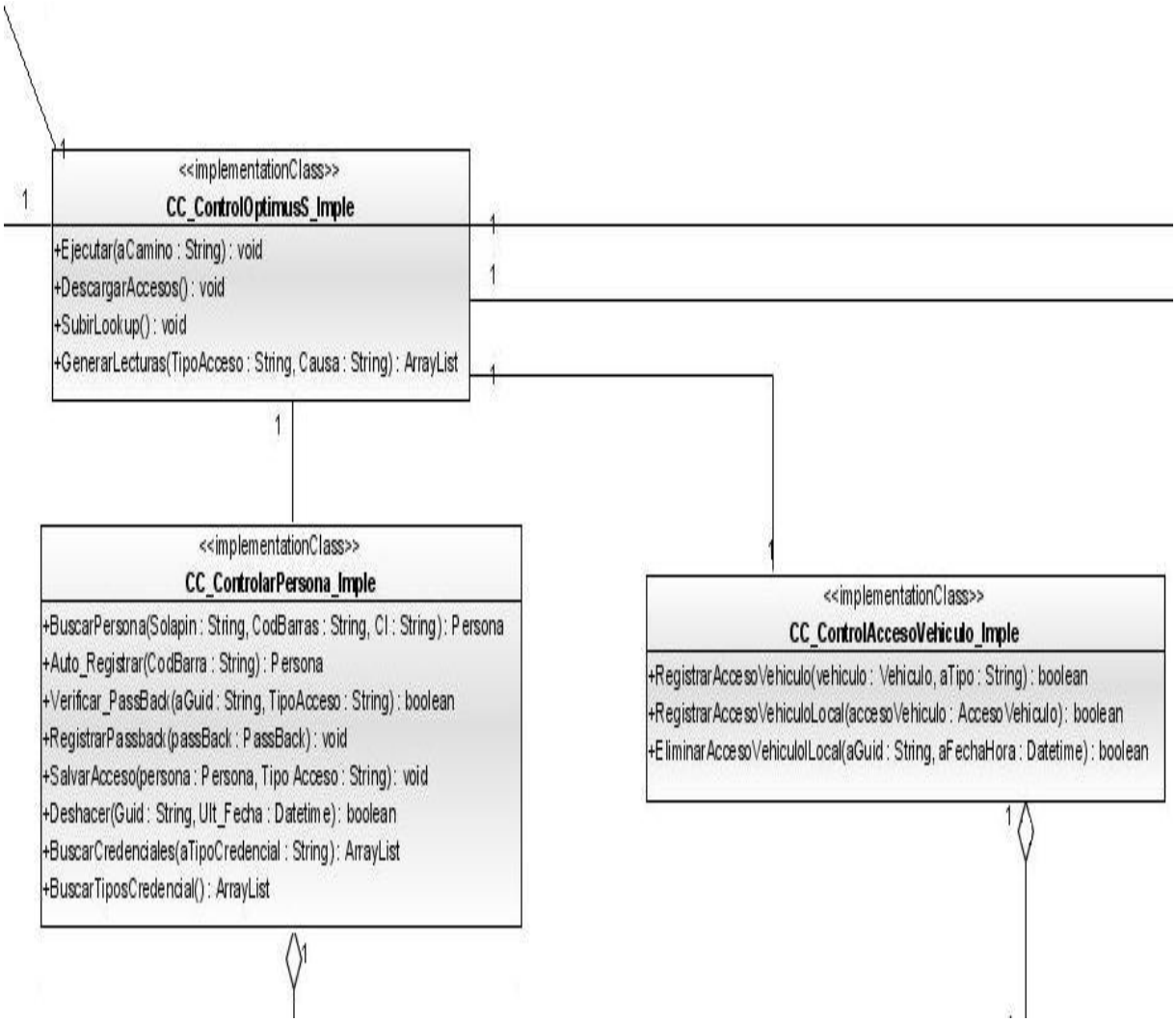


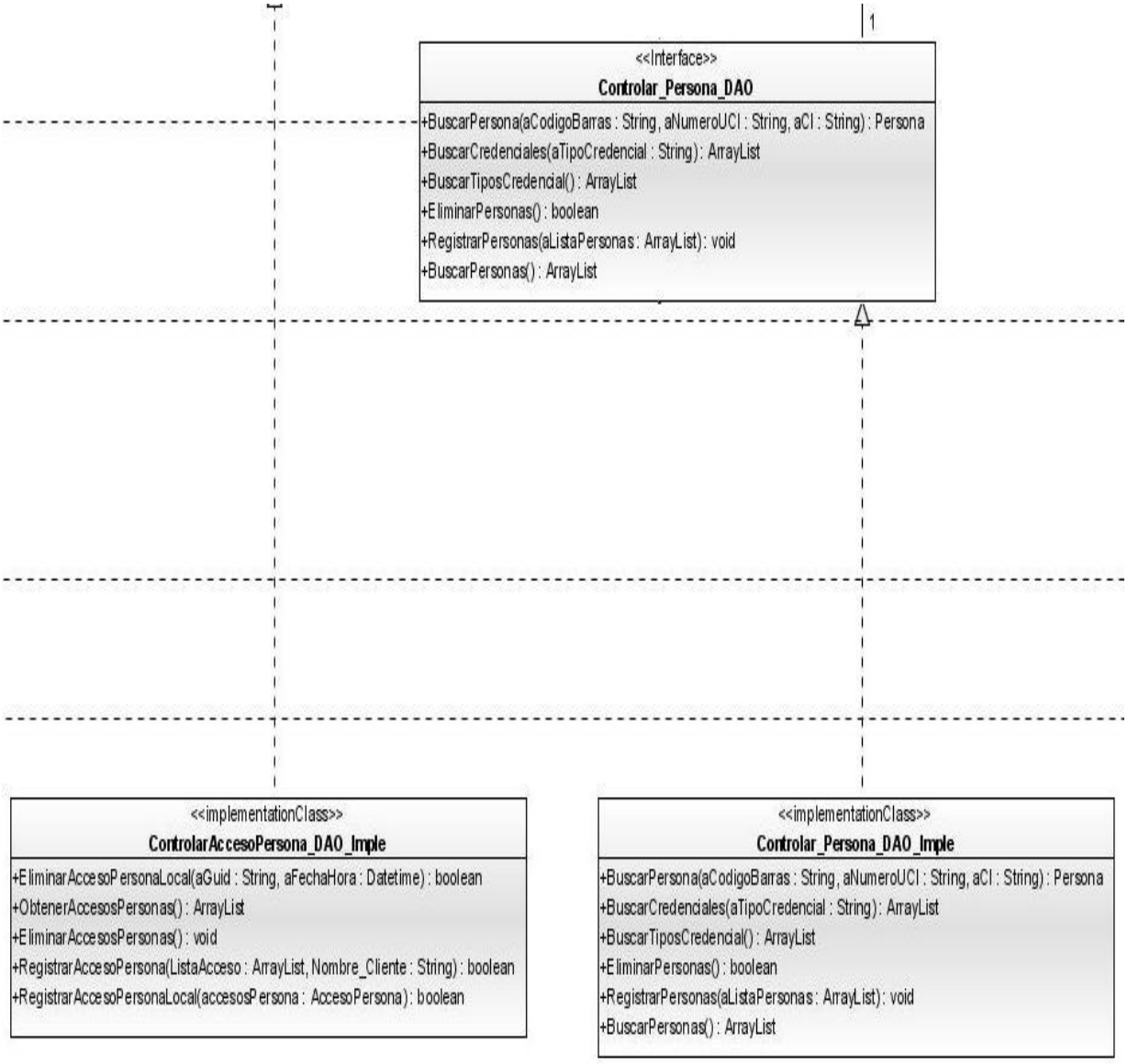
Figura 12 Diagrama de clase del diseño “Actualizar Lector”

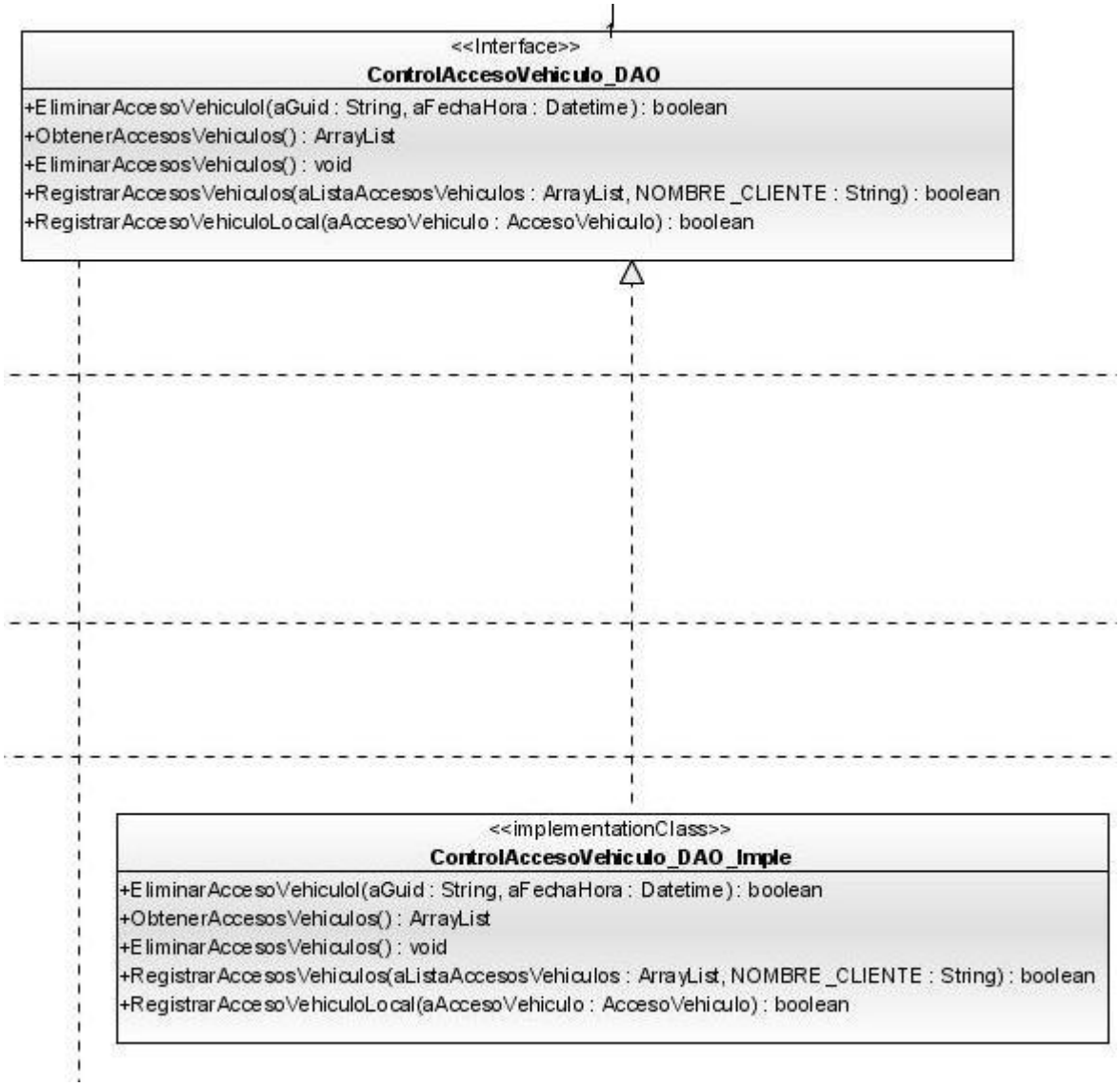

<<Form>>
CI_Actualizar_Base_Dato_Local
-combox_CausaAcceso : jcombobox
-cbox_Personas : jcheckbox
-cbox_Vehiculos : jcheckbox
-btn_Entrada : jbutton
-btn_Salidas : jbutton
+jbutton_Entradas() : void
+jbutton_Salidas() : void
+ValidarVaciaSeleccion() : void
+CargarCausaAcceso(CausasAccesos : ArrayList) : void
1
<<Entity Bean>>
Persona
-PersonasNombreCompleto : String
-CodigoBarras : String
-Solapin : String
-NombreArea : String
-NombreFoto : String
-TipoCredencial : String
-EstadoCredencial : String
-Guid : String
-FechaExpiracion : Datetime
+CPersona()
1
1
<<Entity Bean>>
CausaAcceso
-Id_Causa : int
-Causa : String
1

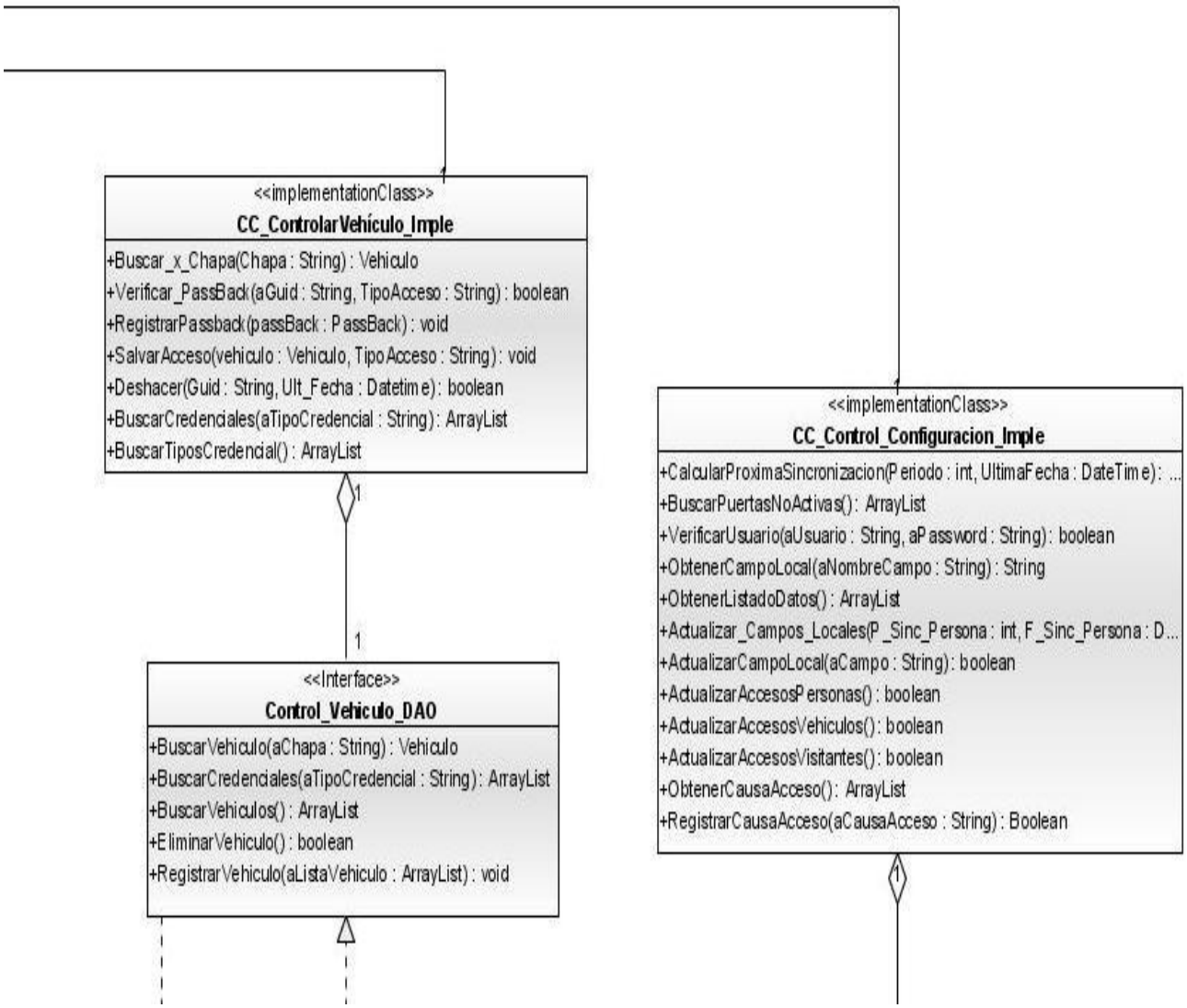

<<Entity Bean>>
ConfigurarSistema
-P_Sinc_Persona : int
-P_Sinc_Vehiculo : int
-P_Sinc_Visitante : int
-P_Sinc_Accesos_Personas : int
-P_Sinc_Accesos_Vehiculos : int
-P_Sinc_Accesos_Visitantes : int
-UF_Sinc_Accesos_Personas : Datetime
-UF_Sinc_Accesos_Vehiculos : Datetime
-UF_Sinc_Accesos_Visitantes : Datetime
-F_Sinc_Persona : Datetime
-F_Sinc_Vehiculo : Datetime
-F_Sinc_Visitante : Datetime
-UF_Sinc_Personas : Datetime
-UF_Sinc_Vehiculos : Datetime
-UF_Sinc_Visitantes : Datetime
-UF_Acceso_Persona : Datetime
-UF_Acceso_Vehiculo : Datetime
-UF_Acceso_Visitante : Datetime
-SN_Sinc_Personas : String
-SN_Sinc_Vehiculos : String
-SN_Sinc_Visitante : String
-U_Guid_Personas : String
-U_Guid_Vehiculos : String
-Ult_CI : String
-UF_Inicio_APP : Datetime
-ID_Puerta : String
-U_Apagado : String
<<Entity Bean>>
Vehiculo
-Chapa : String
-Color : String
-Marca : String
-Tipo_Vehiculo : int
1
0..*
0..*
<<Entity Bean>>
AccesoVehiculo
-Id_Acceso : int
-Chapa : String
-FechaHora : String
-Tipo : String
-GuidChofer


<<Interface>>
CC_Actualizar_Base_Datos_Local
+DescargarAccesos() : void
+GenerarLecturas(TipoAcceso : String, Causa : String) : boolean
+RegistrarCausaAcceso(causaAcceso : String) : boolean
+ObtenerCausaAcceso() : ArrayList
+CargarCausaAcceso(CausasAccesos : ArrayList) : void
<<implementationClass>>
CC_Actualizar_Base_Datos_Local_Imple
+DescargarAccesos() : void
+GenerarLecturas(TipoAcceso : String, Causa : String) : boolean
+ObtenerCausaAceso() : ArrayList
+RegistrarCausaAcceso(causaAcceso : String) : boolean
+CargarCausaAcceso(CausasAccesos : ArrayList) : void
<<implementationClass>>
CC_ControlAccesoPersona_Imple
+RegistrarAccesoPersona(persona : Persona, aTipo : String) : boolean
+RegistrarAccesoPersonaLocal(accesoPersona : AccesoPersona) : boolean
+EliminarAccesoPersonaLocal(aGuid : String, aFechaHora : Datetime) : boolean
<<Entity Bean>>
AccesoPersona
-Id_Acceso : int
-GUID : String
-FechaHora : String
-Id_Tipo : int
-Id_Causa : int
<<Interface>>
ControlarAccesoPersona_DAO
+EliminarAccesoPersonaLocal(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosPersonas() : ArrayList
+EliminarAccesosPersonas() : void
+RegistrarAccesoPersona(ListaAcceso : ArrayList, Nombre_Cliente : String) : boolean
+RegistrarAccesoPersonaLocal(accesosPersona : AccesoPersona) : boolean
1
1
1
1
1
1
0..*
0..*


<<implementationClass>>
CC_ControlOptimusS_Imple
+Ejecutar(aCamino : String) : void
+DescargarAccesos() : void
+SubirLookup() : void
+GenerarLecturas(TipoAcceso : String, Causa : String) : ArrayList
<<implementationClass>>
CC_ControlarPersona_Imple
+BuscarPersona(Solapin : String, CodBarras : String, CI : String) : Persona
+Auto_Registrar(CodBarra : String) : Persona
+Verificar_PassBack(aGuid : String, TipoAcceso : String) : boolean
+RegistrarPassback(passBack : PassBack) : void
+SalvarAcceso(persona : Persona, Tipo Acceso : String) : void
+Deshacer(Guid : String, Ult_Fecha : Datetime) : boolean
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarTiposCredencial() : ArrayList
<<implementationClass>>
CC_ControlAccesoVehiculo_Imple
+RegistrarAccesoVehiculo(vehiculo : Vehiculo, aTipo : String) : boolean
+RegistrarAccesoVehiculoLocal(accesoVehiculo : AccesoVehiculo) : boolean
+EliminarAccesoVehiculoILocal(aGuid : String, aFechaHora : Datetime) : boolean


1
<<Interface>>
Controlar_Persona_DAO
+BuscarPersona(aCodigoBarras : String, aNumeroUCI : String, aCI : String) : Persona
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarTiposCredencial() : ArrayList
+EliminarPersonas() : boolean
+RegistrarPersonas(aListaPersonas : ArrayList) : void
+BuscarPersonas() : ArrayList
<<implementationClass>>
ControlarAccesoPersona_DAO_Imple
+EliminarAccesoPersonaLocal(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosPersonas() : ArrayList
+EliminarAccesosPersonas() : void
+RegistrarAccesoPersona(ListaAcceso : ArrayList, Nombre_Cliente : String) : boolean
+RegistrarAccesoPersonaLocal(accesosPersona : AccesoPersona) : boolean
<<implementationClass>>
Controlar_Persona_DAO_Imple
+BuscarPersona(aCodigoBarras : String, aNumeroUCI : String, aCI : String) : Persona
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarTiposCredencial() : ArrayList
+EliminarPersonas() : boolean
+RegistrarPersonas(aListaPersonas : ArrayList) : void
+BuscarPersonas() : ArrayList


<<Interface>>
ControlAccesoVehiculo_DAO
+EliminarAccesoVehiculoI(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosVehiculos() : ArrayList
+EliminarAccesosVehiculos() : void
+RegistrarAccesosVehiculos(aListaAccesosVehiculos : ArrayList, NOMBRE_CLIENTE : String) : boolean
+RegistrarAccesoVehiculoLocal(aAccesoVehiculo : AccesoVehiculo) : boolean
<<implementationClass>>
ControlAccesoVehiculo_DAO_Imple
+EliminarAccesoVehiculoI(aGuid : String, aFechaHora : Datetime) : boolean
+ObtenerAccesosVehiculos() : ArrayList
+EliminarAccesosVehiculos() : void
+RegistrarAccesosVehiculos(aListaAccesosVehiculos : ArrayList, NOMBRE_CLIENTE : String) : boolean
+RegistrarAccesoVehiculoLocal(aAccesoVehiculo : AccesoVehiculo) : boolean


<<implementationClass>>
CC_ControlarVehículo_Imple
+Buscar_x_Chapa(Chapa : String) : Vehiculo
+Verificar_PassBack(aGuid : String, TipoAcceso : String) : boolean
+RegistrarPassback(passBack : PassBack) : void
+SalvarAcceso(vehiculo : Vehiculo, TipoAcceso : String) : void
+Deshacer(Guid : String, Ult_Fecha : Datetime) : boolean
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarTiposCredencial() : ArrayList
1
1
<<Interface>>
Control_Vehiculo_DAO
+BuscarVehiculo(aChapa : String) : Vehiculo
+BuscarCredenciales(aTipoCredencial : String) : ArrayList
+BuscarVehiculos() : ArrayList
+EliminarVehiculo() : boolean
+RegistrarVehiculo(aListaVehiculo : ArrayList) : void
<<implementationClass>>
CC_Control_Configuracion_Imple
+CalcularProximaSincronizacion(Periodo : int, UltimaFecha : DateTime) : ...
+BuscarPuertasNoActivas() : ArrayList
+VerificarUsuario(aUsuario : String, aPassword : String) : boolean
+ObtenerCampoLocal(aNombreCampo : String) : String
+ObtenerListadoDatos() : ArrayList
+Actualizar_Campos_Locales(P_Sinc_Persona : int, F_Sinc_Persona : D...
+ActualizarCampoLocal(aCampo : String) : boolean
+ActualizarAccesosPersonas() : boolean
+ActualizarAccesosVehiculos() : boolean
+ActualizarAccesosVisitantes() : boolean
+ObtenerCausaAcceso() : ArrayList
+RegistrarCausaAcceso(aCausaAcceso : String) : Boolean
1

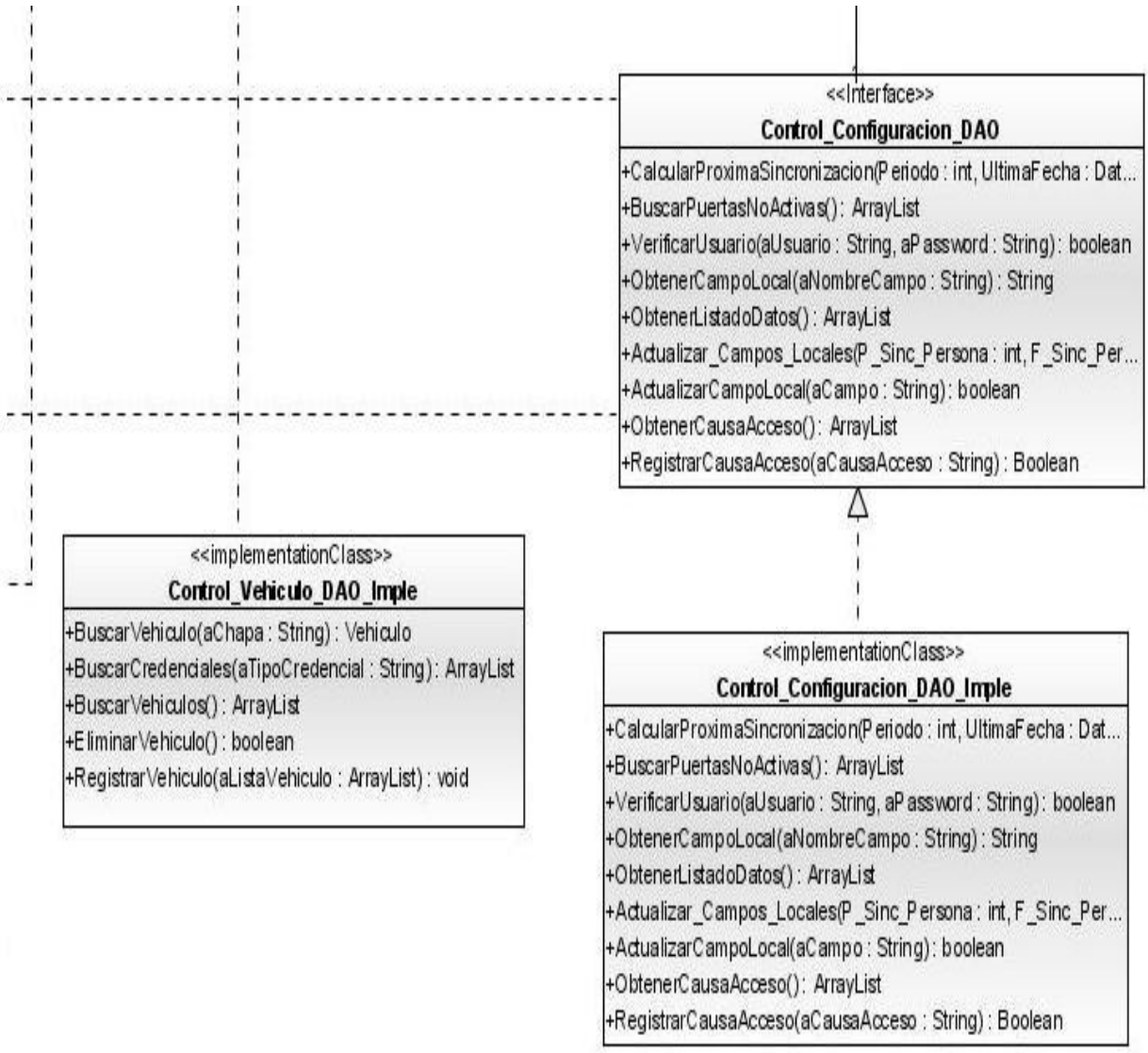


Figura 13 Diagrama de clase del diseño "Actualizar Base de Datos Local