Universidad del Azuay

Facultad de Ciencias de la Administración

Escuela de Ingeniería de Sistemas

Desarrollo de Interfaces para equipos de Identificación por Radio Frecuencia (RFID), aplicado al control de ingreso de estudiantes a los laboratorios.

Tesis previa a la obtención del título de Ingeniero en Sistemas.

Autores: Palacios Andrade Jessica Priscila
Rodas Lema Paúl Adrián

Director: Ing. Pablo Esquivel

Cuenca, Ecuador

2012

Agradecimientos

Queremos agradecer a Dios por ser nuestro guía en este largo camino recorrido.

A nuestros Padres y Hermanos por todo el apoyo incondicional en este trabajo y en nuestra vida, por darnos una formación académica y sobre todo humanista y espiritual.

Y un agradecimiento especial al Ing. Pablo Esquivel por su gran apoyo y motivación para la culminación de nuestros estudios profesionales y para la elaboración de esta tesis.

Índice de Contenidos

Índice de ilustraciones y cuadros

Índice de Anexos

Resumen

La Identificación por Radio Frecuencia RFID es, sin duda, una de las tecnologías de captura de datos que ha experimentado un crecimiento acelerado en los últimos tiempos y la falta de uso por la dificultad de implementación de la tecnología, nos indujo a desarrollar el presente trabajo que tiene como propósito, permitir a un mayor rango de aplicaciones utilizar RFID, mediante una interfaz configurable, tanto en comportamiento como en persistencia de los datos capturados, para aplicaciones compatibles con bases de datos o archivos XML, demostrando, con un ejemplo las sustanciales ventajas en varios ámbitos de aplicación.

ABSTRACT

Radio-frequency identification RFID, without doubt, is one of the technologies for capturing data that has experienced an accelerated development in the last years. The lack of its use due to the difficulty to implement this type of technology led us to develop the present research. The purpose is to allow a higher range of applications to employ RFID through a configurable interface of both the behavior and persistency of the captured data. This will be employed with applications that are compatible with XML database and we will be able demonstrate the substantial advantages in several areas of the application through a practical example.

UNIVERSIDAD DEL
AZUAY
DPTO. IDIOMAS

Translated by,
Diana Lee Rodas

Certificado de Autoría

Nosotros, Palacios Andrade Jessica Priscila y Rodas Lema Paúl Adrián, certificamos que la tesis denominada, "Desarrollo de *Interfaces* para equipos de Identificación por Radio Frecuencia (RFID), aplicado al control de ingreso de estudiantes a los laboratorios", la cual presentamos como requisito para el grado de Ingenieros en Sistemas de la Universidad del Azuay, es de exclusiva responsabilidad de los autores.

Así mismo, damos fe de que las ideas, código fuente y resultados vertidos en ésta tesis son originales e inéditos.

Jessica Palacios Andrade
010415963-7

Paúl Rodas Lema
010525302-5

Dedicatoria

A Dios.
Por habernos permitido llegar hasta este punto y habernos dado salud para lograr nuestros objetivos, además de su infinita bondad y amor.

A nuestros Padres.
Por ser el pilar fundamental en todo lo que somos, en toda nuestra educación, tanto académica, como de la vida, por su apoyo, consejos y valores, por la motivación constante que nos ha permitido ser una persona de bien, pero más que nada, por su amor.

A mi Esposo.
Adrián Bravo por estar conmigo en aquellos momentos en que el estudio y el trabajo ocuparon mi tiempo y esfuerzo.

Jessica.

Objetivo General:

Desarrollar una interfaz configurable para el intercambio de datos con otras aplicaciones utilizando las API de la empresa ALIEN para productos RFID.

Objetivos Específicos:

- Construir una interfaz para la comunicación con las API de equipos RFID de marca ALIEN.

- Crear un manual de las diferentes API utilizadas para el desarrollo de las aplicaciones.

- Desarrollar software utilizando API de RFID Alien que permita la el intercambio de datos con otras aplicaciones, control de mensajes y errores, y la configuración de procesamiento de las etiquetas por lotes o por tiempos.

- Construir un aplicativo que identifique a los estudiantes que ingresan a los laboratorios de computación.

CAPITULO I. Tecnología RFID (Identificación por Radio Frecuencia)

1.1 Introducción RFID

La tecnología RFID (*Radio Frequency IDentification*, en español Identificación por Radio Frecuencia) es una herramienta muy valiosa para los negocios, se prevé que en un fututo remplace las tecnologías de identificación existentes como el código de barras.

En términos generales la tecnología RFID permite la identificación de objetos de forma inalámbrica, sin necesidad de que exista entre el lector y el objeto contacto o línea de visión directa, requisito indispensable para otras tecnologías.

El presente trabajo pretende proporcionar conocimiento suficiente para entender los sistemas RFID, su surgimiento, arquitectura, estándares a los que se rige a nivel mundial como las instituciones que las norman, los tipos de dispositivos que se utilizan y sus beneficios potenciales.

1.2 Reseña Histórica RFID

No existe información muy certera acerca de los orígenes de RFID como tal, posiblemente debido a que es la convergencia de varios principios que fueron evolucionando con el tiempo y haciendo factible esta tecnología como la conocemos en la actualidad, sin embargo, al investigar encontramos algunos datos.

Se ha sugerido que el primer dispositivo conocido similar a RFID pudo haber sido una herramienta de espionaje inventada por Léon Theremin para el gobierno soviético en 1945. El dispositivo de Theremin era un dispositivo de escucha secreto pasivo, no una etiqueta de identificación, por lo que esta aplicación es dudosa. Según algunas fuentes, la tecnología usada en RFID habría existido desde comienzos de los años 1920, desarrollada por el MIT (*Massachusetts Institute of Technology*) y usada extensivamente por los británicos en la Segunda Guerra Mundial (fuente que establece que los *sistemas* RFID han existido desde finales de los años 1960 y que sólo recientemente se había popularizado gracias a las reducciones de costos)...

Una tecnología similar, el transpondedor o *tag* de IFF (Identificador Amigo-Enemigo), fue inventada por los británicos en 1939, y fue utilizada de forma rutinaria

por los aliados en la Segunda Guerra Mundial para identificar a los aeroplanos como amigos o enemigos. (RFID Pasaporte Electrónico, 2005)

Otro trabajo temprano que trata el RFID es el artículo de 1948 de Harry Stockman, titulado "Comunicación por medio de la energía reflejada". Stockman predijo que "... el trabajo considerable de investigación y de desarrollo tiene que ser realizado antes de que los problemas básicos restantes en la comunicación de la energía reflejada se solucionen, y antes de que el campo de aplicaciones útiles se explore." Hicieron falta treinta años de avances en multitud de campos diversos antes de que RFID se convirtiera en una realidad. (Actas del IRE, pp. 1196-1204, octubre de 1948)

A partir de 1980, se focaliza en la comercialización de la RFID, desde nuevas aplicaciones a mejoras en el comportamiento y el coste de lectores, etiquetas y antenas. El éxito es evidente viendo la situación actual de la RFID.
En el 2000, *Auto-ID Center* focaliza todos sus esfuerzos en el desarrollo tecnológico para la implantación masiva de la tecnología RFID en la cadena de suministro, proporcionando un sustituto al código de barras. Posteriormente se convierte en EPC Global para gestionar y desarrollar estándares.

A pesar de que ya existen muchas aplicaciones prácticas en el mundo empresarial, es posible que los próximos años tenga un sorprendente desarrollo debido a la amplitud de posibilidades y la eficacia de sus resultados, principalmente en la logística y trazabilidad brindado un mundo infinito de opciones.

1.3 Definición RFID

Existen varias definiciones para RFID (Identificación por Radio Frecuencia), la siguiente definición es simple y precisa:

"RFID es un modo automático para recolectar datos de productos, lugar, tiempo o transacciones rápida y fácilmente sin intervención humana o error".

RFID es un método remoto de almacenamiento y recuperación de datos que usa dispositivos denominados lector, antena, *tags* o etiquetas que transportan los datos. El lector transmite una señal de radio mediante su antena, la cual es recibida por la

etiqueta a través de su propia antena y usada para potenciar un circuito integrado. Con la energía que obtiene de la señal cuando ingresa al campo radal, el *tag* conversará brevemente con el lector para verificar e intercambiar datos. Una vez que el lector recibe los datos, luego se envían a una computadora controlada para su procesamiento y gestión.

(AIMglobal Asociacion de Tecnologias de Identificacion Automatica y Captura de Datos, 2006)

1.4 Códigos EPC

1.4.1 Definición Códigos EPC

EPC (*Electronic Product Code* o en español Código Electrónico de Producto), es la evolución del código de barras ya que, al contrario de éste no necesita ser manipulado para su lectura.

El EPC es un número único que se encuentra almacenado en un *tag* de radiofrecuencia, que permite identificar cada producto de manera única. Así mismo arroja información valiosa como la posibilidad de conocer, en cualquier momento, la ubicación física del producto. Con un *tag* es posible conocer información detallada de los productos, como por ejemplo longitud o grosor, fechas, lugar de fabricación y ubicación.

El EPC está diseñado para incluir distintos sistemas de codificación. El más utilizado es el Sistema de Codificación Estándar GS1, administrados a nivel mundial por EPCglobal.

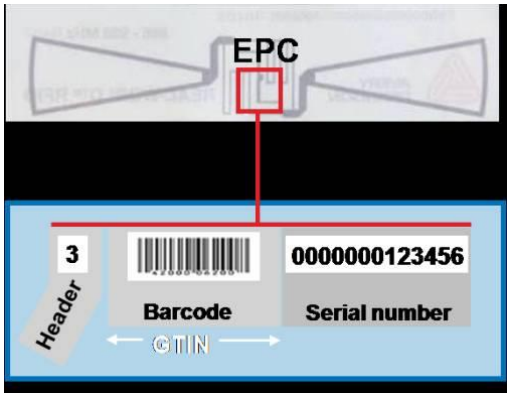


Gráfico 1. Estructura de los Códigos EPC

La estructura del EPC, básicamente unifica el código de barras y el número de serie, de esta manera se puede identificar ya no solo un producto en particular sino también el número de serie específico, por lo que cada producto es individual y único. (GS1, 1992)

1.4.2 Funcionalidad del Código EPC

El Código Electrónico de Producto es utilizado como un único identificador de artículo y un puntero a la información.

- **Como identificador único:** El EPC identifica a cada artículo, ya que contiene una serie (un número único asignado a cada ocurrencia del artículo). A modo de ejemplo, un pallet del artículo comercial A tendrá un código EPC distinto de otro pallet del mismo artículo comercial A. El Sistema GS1 garantiza que cada empresa tenga su identificación de manera única a nivel mundial.

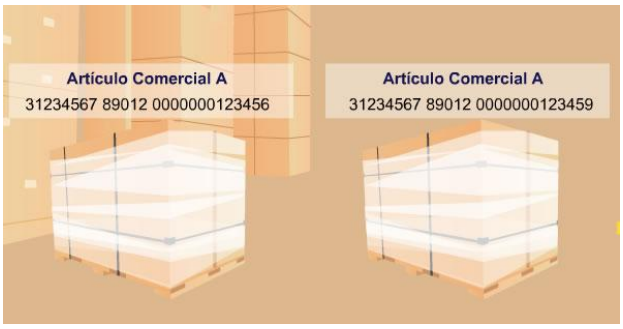


Gráfico 2. EPC como Identificador Único

- **Como Puntero de Información:** El EPC es un elemento clave para la información contenida en los sistemas TI. Esto permite la recuperación de la información del artículo. La información puede ser sobre los movimientos del artículo o la producción estática de datos relacionados o los datos maestros**.**

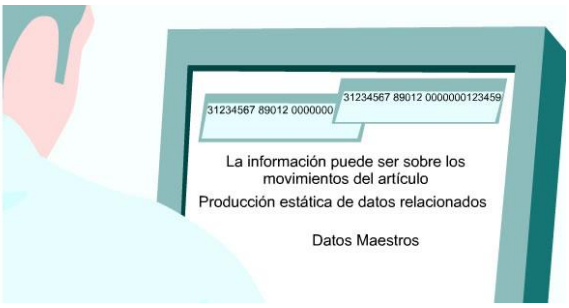


Gráfico 3. EPC como Puntero de Información

(GS1 EPC, 1992)

1.4.3 Formato del Código EPC

El formato general para los datos de la etiqueta EPC incluyen las siguientes secciones:

- **Encabezado (*Header*):** El encabezado identifica la versión numérica del código por sí mismo.
- **Administrador EPC:** Identifica una empresa que es responsable de mantener la Categoría de objeto y Número serial. EPC Global asigna el Administrador General a una entidad, asegurando que cada uno de estos números sea único.
- **Categoría de Objeto:** Se refiere al tipo exacto de producto, similar a un SKU (unidad mínima de producto). La Categoría de Objeto es usado por una entidad de gestión EPC para identificar ítems de mercado. Estos números de categoría de Objeto, por supuesto, deben ser únicos dentro de cada dominio del Número de Administrador General. El ejemplo más común de Categoría de Objeto seria el SKU de bienes de consumo.
- **Número Serial:** Representa un único identificador para el ítem dentro de cada categoría de objeto. La entidad administradora es responsable por la asignación unívoca de números seriales no repetitivos para cada instancia dentro de cada categoría de objeto.

Gráfico 4. Formato del Código EPC

Como se puede observar en el Gráfico 4, el ECP nos permite definir:

1) Versión del EPC utilizada, 8 bits, 256 Opciones.
2) Identificación del fabricante, 28 bits, 268 millones de Fabricantes.
3) Tipo de producto, 24 bits, 16 millones de clases de productos
4) Número de serie ÚNICO del objeto, 36 bits, 68 millones de números de serie.

Este número de combinaciones resulta inimaginable y para que se lo pueda llevar a la práctica se tiene que esperar el perfeccionamiento del RFID.
(RFIDpoint, 2005)

1.4.4 Instituto regulador EPCglobal

EPC Global Inc. es una organización independiente, sin fines de lucro y con estándares globales encomendados por la industria para el manejo de la adopción e implementación de la Red EPC Global y la tecnología EPC.
Es líder en la elaboración de las normas establecidas por la industria para el Código Electrónico de Producto para apoyar el uso de la Identificación por Radio Frecuencia.
La Red EPC Global es un sistema que permite el intercambio de información entre socios comerciales. La información se captura de la *tag* EPC y se la almacena en la base de datos de la Red EPC. La Red ofrece información en tiempo real sobre los artículos EPC que se mueven dentro de la cadena de abastecimiento desde el fabricante hasta el consumidor final pues utiliza el internet como principal vía de comunicación.

1.4.5 Clasificación del Código EPC

La estructura del EPC se acopla a diferentes usos prácticos del código, los más relevantes los describimos a continuación:

- **SGTIN**: (*Serialized Global Trade Item Number*) Número de serie global, utilizado para la identificación de productos, por fabricante, clase de producto y número de serie específico del mismo. Uso industrial.
- **SSCC:** (*Serial Shipping Container Code*) Código de serie de contenedor, también tiene un uso industrial, pero identifica las diferentes presentaciones de empaque de un mismo producto.
- **SGLN:** (*Serial Global Location Number*), Numero de serie del posicionamiento global.

- **GRAI: (***Global Returnable Asset Identifier*), Identificador Global de Activos retornables.

(EPC, 1992)

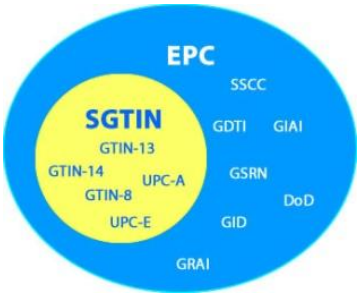


Gráfico 5. Clasificación Código EPC

1.5 Arquitectura RFID

El lector envía una señal de radio que es recibida por todos los *tags* presentes en el campo de radiofrecuencia sintonizado con dicha frecuencia. Los *tags* reciben la señal a través de sus antenas y responden transmitiendo los datos que almacenan. El *tag* puede almacenar muchos tipos de datos, como el número de serie, instrucciones de configuración, historial de actividad (por ejemplo, fecha del último mantenimiento, paso del *tag* por una ubicación concreta, etc.) o incluso la temperatura y otros datos proporcionados por los sensores. El dispositivo de lectura/escritura recibe la señal del *tag* a través de su antena, la descodifica y transfiere los datos al sistema informático a través de una conexión de cable o inalámbrica.

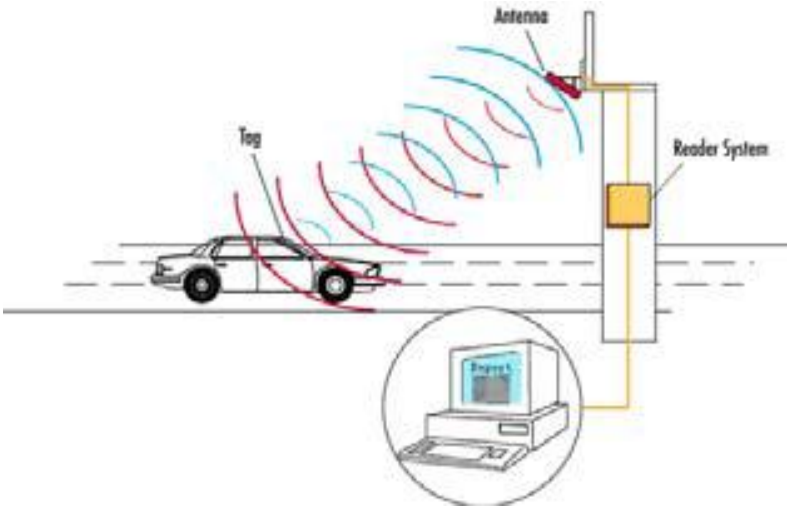


Gráfico 6. Componentes Sistema RFID

A continuación los componentes de un sistema RFID:

1. Etiqueta RFID, transpondedor o *tags*

2. Lector de RFID o transceptor
3. Antenas Receptoras
4. Subsistema de procesamiento de datos o Middleware RFID

(Nodus, 1999)

1.5.1 Etiqueta o *Tag* RFID

Cuando el lector transmite en el espacio, espera normalmente una respuesta de otro elemento para mantener la comunicación, en los sistemas RFID es el *Tag* quien responde.

1.5.1.1 Componentes de las *Tags*

Los *tags* RFID constan de dos elementos básicos: un chip y una antena. El chip y la antena, montados, forman un integrado.

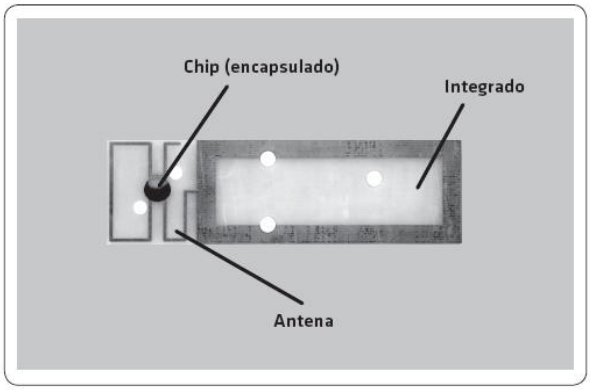


Gráfico 7. Componentes de la *Tag*

- **Chip o circuito integrado:** el chip almacena la información y ejecuta los comandos específicos. La mayoría de los *tags* pasivos que deben cumplir solo con la misión de matrícula de producto tiene 96 bits (como el EPC), pero pueden tener una capacidad mayor. Como se puede entender a mayor capacidad mayor es el coste de producción. El diseño del chip determina el tipo de memoria, si es de solo lectura o tiene la capacidad de leer y escribir.

- **Antena:** La función de la antena es absorber las ondas RF y entonces difundir por el mismo medio la información contenida en el chip. La energía para activar el chip la colecta del campo RF (en HF del campo electromagnético y

en UHF del campo eléctrico). Este proceso es llamado acoplamiento (*coupling*). En términos más técnicos un *coupling* describe cuando la energía se transfiere de un sistema a otro, en nuestro caso del aire a la antena. El tamaño de la antena es crítico para el comportamiento del *tag* porque normalmente determina el rango de lectura del *tag*. Simplemente al poner una antena más grande, esta puede recolectar mayor energía y por lo tanto puede trasmitir con más potencia.

Otras características de las antenas es la frecuencia de emisión y recepción, podemos encontrarnos con *Low Frecuency* (LF) y *High Frecuency* (HF) donde las antenas son espirales por ser frecuencia magnéticas en la natura, o *Ultra High Frecuency* (UHF) más puramente eléctricas. El tamaño también afecta a la frecuencia de emisión recepción.

(Invenligent, 2007)

1.5.1.2 Clasificación de las *Tags*

Las tags se clasifican por:

- **Tipo de Memoria:**
  - **RO (*Read Only* o Solo Lectura):** como indica su nombre solo de lectura, el identificador viene gravado de fábrica y tiene una longitud fija de caracteres.
  - **WORM (*Write Once Read Many* o Escritas una vez y múltiples lecturas):** programable por el usuario una unidad de escritura, pudiendo leer las veces que se quiera.
  - **Escritura y lectura múltiples:** una parte de la memoria, normalmente de usuario, se puede grabar hasta 100.000 veces. Estos *tags* se utilizan para aplicaciones cerradas de la misma empresa y necesitan reutilización de los *tags*.

- **Clase**

  La EPC global como órgano de estandarización para la RFID, en su uso con EPC ha organizado las etiquetas en 6 clases.

  - **Clase 0:** solo lectura (el número EPC se codifica en la etiqueta durante el proceso de fabricación).

- **Clase 1:** escritura una sola vez y lecturas indefinidas (se fabrican sin número y se incorpora a la etiqueta más tarde)
- **Clase 2:** lectura y escritura.
- **Clase 3:** capacidades de la clase 2 más la fuente de alimentación que proporciona un
- incremento en el rango y funcionalidades avanzadas.
- **Clase 4:** capacidades de la clase 3 más una comunicación activa con la posibilidad de comunicar con otras etiquetas activas.
- **Clase 5:** capacidades de la clase 4 más la posibilidad de poder comunicar también a etiquetas pasivas.

(RFIDsolutions, 2008)

❖ **Fuente de energía:**

- **Activos:** precisan de mayor energía, la cual es suministrada por baterías. Dicha energía es utilizada para activar la circuitería del microchip y enviar la señal a la antena. Permiten una amplia cobertura de difusión, es decir, mayor alcance. Normalmente tienen una mayor capacidad de almacenar información, más allá del simple código único, como el contenido, el origen, destino, procesos realizados, etc. También pueden llevar sensores adicionales a la propia memoria como sensores de temperatura, de velocidad, de movimiento, etc. que permiten almacenar o controlar datos vitales en algunas aplicaciones.

  Estos *tags* son los más caros del mercado pero tienen un retorno de la inversión en muchas aplicaciones.

  Los Ministerios de Defensa, por ejemplo, identifican los *containers* mediante esta tecnología para saber entre muchas otras cosas, el contenido exacto de su interior. También etiquetan elementos muy caros para su gestión de activos. Otro ejemplo, es su utilización en aplicaciones ferroviarias, donde se pueden integrar con sistemas GPS. El ejemplo más claro de *tags* activos es el sistema TeleTac para el pago sin parar de peajes.

Gráfico 8. *Tags* Activos

- **Pasivos:** no requieren batería ya que toda la energía la recoge del campo electromagnético creado por el lector. Como es de suponer son los más económicos y los de menor rango de comunicación, pero por su relación entre comportamiento y precio son los más utilizados.

  Sin embargo es importante entender que tienen menor cobertura en el rango de lectura, requieren de la energía del campo electromagnético del lector, por lo que requieren de mayor aproximación a este campo, por lo que limitaría a una distancia entre cinco y diez metros dependiendo de la antena de recepción.

  Su uso se da principalmente en los grandes minoristas como WalMart y el departamento de defensa de los Estados Unidos, son de fácil producción en volumen y de bajo costo.

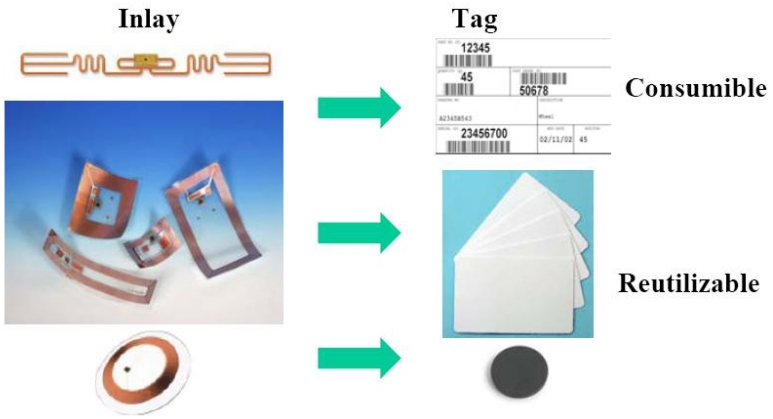


Gráfico 9. *Tags* Pasivos

- **Semi Pasivo:** utiliza una batería para activar la circuitería del chip pero la energía para generar la comunicación es la que recoge de las ondas radio del lector (como en los pasivos).

  Debido a la utilización de batería, estos son más grandes y caros que los pasivos, pero consiguen mejores rangos de comunicación. Algunos *tags* llevan integrados sensores de temperatura, movimiento, etc. para proporcionar mayores funcionalidades.

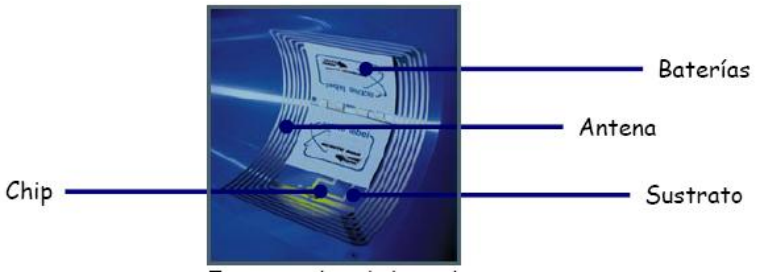


Gráfico 10. *Tags* Semi Pasivos

(Nodus, 1999)

1.5.2 Lector RFID

El principal objetivo de un lector de RFID es transmitir y recibir señales, convirtiendo las ondas de radio de los *tags* en un formato legible para las computadoras.

Gráfico 11. Lector RFID de Alien Technology

En su fabricación se suelen separar en dos tipos:

- **Sistemas con bobina simple**, la misma bobina sirve para transmitir la energía y los datos. Son más simples y más baratos, pero tienen menos alcance.
- **Sistemas interrogadores con dos bobinas**, una para transmitir energía y otra para transmitir datos. Son más caros, pero consiguen mayores prestaciones.

Compuesto por una antena, un transceptor y un decodificador. No solo genera la señal que a través de las antenas se transmite en el aire, sino que también escucha las respuestas de las *tags*, transmite y recibe ondas analógicas que transforma en cadenas de bits de 0 y 1, bits de información digital, cada lector es conectado a una o más antenas (máximo según tipo de lectores), estas tienen una ciencia propia, pero es importante conocer como el lector crea la señal electromagnética y la antena realiza la difusión en su zona de interrogación (campo de radio frecuencia), además el lector también se conecta a la red o a una máquina mediante varios tipos de interfaz como pueden ser RS-232 o Ethernet, a veces cuando se habla de lectores, ya se entiende que también se habla de las antenas, ya que existen lectores con antenas integradas y otros que necesitan su conexión.

Hay varios tipos de lectores: simples (un solo estándar y frecuencia), multi regionales, multi frecuencias (trabajan a diferentes frecuencias), multi protocolos, etc. la línea con mayor interés son los lectores ágiles y flexibles que pueden utilizar cualquier protocolo, región o frecuencia (HF o UHF) según su uso.
(LectoresRFID, 2006)

1.5.2.1 Factores para escoger un Lector

El núcleo de los lectores RFID es, en esencia, un transcriptor de señales de radio que, al mismo tiempo, transmite y recibe señales de radio con el *tag* RFID. Lo que ello nos indica es que un lector RFID tiene que hacer frente a una combinación de retos tecnológicos habituales de los sistemas de radio con otros retos no habituales, que son típicos de la comunicación *wireless*, pero bien conocidos por los técnicos en radares y expertos en comunicaciones pasivas, como es el caso del RFID.

Los lectores RFID, además de tener como características la Exactitud, la Eficiencia, la Flexibilidad con un bajo Ruido de radiación, deberán de tenerse muy en cuenta 6

Factores fundamentales y prácticos que nos ayudarán a escoger el lector RFID adecuado para nuestro trabajo:

1. **Sensibilidad**. Deberá poder detectar señales procedentes del *tag* RFID de hasta -60 dBm de potencia, que es la mínima potencia que le puede llegar de un *tag* RFID. Hoy, es posible detectar señales de hasta -115 dBm. Los buenos lectores RFID llegan a -80 dBm.
2. **Selectividad**. Deberá poder seleccionar la señal procedente del *tag* RFID dentro de un vasto espectro de señales recibidas, algunas mucho más potentes que ella. Este aspecto resulta tan obvio como de vital importancia ya que las frecuencias RFID trabajan cerca de las frecuencias de telefonía y, si no se tiene en cuenta, pueden existir interferencias.
3. **Alcance Dinámico**. Deberá de poder detectar y seleccionar señales procedentes, al mismo tiempo, de varios *tag* RFID que estén a distancias diferentes, con lo que las potencias de emisión del *tag* pueden diferir en un factor mayor de 10.000 de diferencia.
4. **Trabajar bajo Normativas**. En Europa, la normativa RFID permite operar entre 865,6-867,6 MHz de banda de frecuencia, con una potencia máxima del lector RFID de 2 W. En Europa la entidad reguladora es ETSI (European Telecommunications Standard Institute) con la normativa EN 302 208.
5. **Operatividad en entornos Densos de lectores RFID**. Es una norma suplementaria, no obligatoria como una legislación, pero muy útil para poder soportar interferencias con otros lectores RFID. Para estar en conformidad con el estándar EPC Global Gen2 hace falta cumplir con esta norma.
6. **Inter-Operatividad multi-Fabricante**. Es una norma suplementaria, no obligatoria como una legislación, pero muy útil para poder trabajar con todo tipo de fabricantes de chips RFID y lectores RFID siendo intercambiables sus productos sin ningún problema. EPC Global tiene una certificación de inter-operatividad a disposición del mercado. (LectoresRFID, 2006)

1.5.2.2 Clasificación de los Lectores

Encuadrar los lectores RFID dentro de unos parámetros para poder obtener algún tipo de comparación práctica, no resulta un trabajo fácil. Los mismos fabricantes se cuidan de mostrar características técnicas diferentes que sus competidores para evitar ser comparados.

Así, se ha realizado la siguiente clasificación de lectores RFID:

- **Lectores RFID fijos**; son aquellos que salieron en sus principios y se han convertido en el estándar que todos conocemos. Estos tienen 3 subgrupos:
    1. ***Tag-Reader* HP**, que es el lector RFID clásico pero con altas prestaciones, homologado por todo tipo de organismos reguladores en materia de radio y de protocolos con el fin de poder ser utilizados sin ningún inconveniente. Se pueden utilizar para puestos simples y portales multi*Reader* entre 1 y 4 antenas.

Gráfico 12. *Tag-Reader* HP

    2. ***Tag-Reader* LP**. Estos lectores RFID son del mismo tipo de concepto que el *Tag-Reader* HP pero con prestaciones más bajas y precio de adquisición más económico.

Gráfico 13. *Tag-Reader* LP

3. **Lectores RFID 3D**, para obtener una total visibilidad de la cadena para la trazabilidad de los productos, incluyendo activos, materia prima, producto semi-elaborado, producto acabado, expediciones y carga en el muelle.

Gráfico 14. Lectores RFID 3D

- **Lectores RFID Móviles**; son aquellos que nos permiten viajar con ellos tales como en vehículos industriales, carretillas o como dispositivos de lectura manuales. Estos se dividen en **lectores RFID de carretilla** y **lectores RFID manuales**.

Gráfico 15. Lectores Móviles

(Dipole, 2005)

1.5.3 Antenas RFID

Son los dispositivos que permiten radiar las señales de los lectores y leer las ondas radio de los *tags*, como hemos comentado en los lectores, muchas veces se habla de

lector con antena integrada como si fuera un lector (ya se entiende que tiene una antena), varias antenas pueden ser gestionadas por un único lector/grabador, en este caso sí que se distingue bien lo que es el dispositivo lector de la antena o antenas. (AntenasRFID, 2004)

Gráfico 16. Antenas RFID

1.5.3.1 Clases de Antenas

Hay dos clases de antenas, en la mayoría de veces también podemos catalogar los lectores con estos dos tipos:

- **Antenas Móviles:** normalmente se encuentran en lectores móviles con antenas integradas o son utilizadas manualmente por un operario (tipo aspirador de *tags* o buscador de *tags*). En resumen cuando la antena se mueve para identificar el *tag*.

  Estas son especialmente para soluciones de movilidad, como identificación de productos en rutas de ventas, servicio técnico, etc. Que normalmente no requieren la conexión a sistemas informáticos en línea sino más bien captura de información o identificación para accesos.

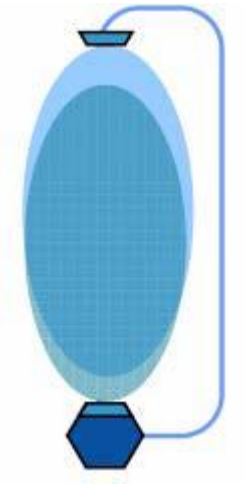
Lector móvil para ranuras compaq flash Lector RFID con conexión para Bluetooth

Gráfico 17. Antenas Móviles

- **Antenas Fijas**: como su nombre indica son antenas fijas que se conectan a los lectores mediante cables, un único lector puede gestionar varias antenas creando una zona de interrogación, podemos encontrar ejemplos en las Dock Door (2 antenas) para puertas o de arco (3 antenas) para cintas transportadoras.

(RFIDsolutions, 2008)

Antenas dock door o arco



Zona de interrogación

Gráfico 18. Antenas Fijas

1.5.4 Subsistema de procesamiento de datos o Middleware RFID

El middleware es el software que se ocupa de la conexión entre el hardware de RFID y los sistemas de información existentes, tal como sistemas de gestión de inventarios, ERPs, CRMs, etc. Su función es la de gestionar todo el sistema RFID a nivel de hardware, recibir la totalidad de la señales de los *tags* y filtrar la información, para solo transmitir información útil a los sistemas empresariales. El middleware también puede ser un software diseñado expresamente para una aplicación concreta, que lo único que haga es transmitir la información recogida por los lectores a la aplicación correspondiente.

Se ocupa, entre otras cosas, del encaminamiento de los datos entre los lectores, las etiquetas y los sistemas de información, y es el responsable de la calidad y usabilidad de las aplicaciones basadas en RFID.
Por ejemplo, en un sistema RFID basado en etiquetas, en el proceso de lectura se ocuparía de la transmisión de los datos almacenados en una de las etiquetas al sistema de información.

Las cuatro funciones principales del middleware de RFID son:

1. **Adquisición de datos.** El middleware es responsable de la extracción, agrupación y filtrado de los datos procedentes de múltiples lectores RFID en un sistema complejo. Sin la existencia del middleware, los sistemas de información de las empresas se colapsarían con rapidez. Por ejemplo, se ha estimado que cuando Walmart empezó a utilizar RFID, generaba del orden de 2 TBytes de datos por segundo.

2. **Encaminamiento de los datos.** El middleware facilita la integración de las redes de elementos y sistemas RFID de la aplicación. Para ello dirige los datos al sistema apropiado dentro de la aplicación.

3. **Gestión de procesos**. El middleware se puede utilizar para disparar eventos en función de las reglas de la organización empresarial donde opera, por ejemplo, envíos no autorizados, bajadas o pérdidas de stock, etc.

4. **Gestión de dispositivos**. El middleware se ocupa también de monitorizar y coordinar los lectores RFID, así como de verificar su estado y operatividad, y posibilita su gestión remota.

(MiddlewareRFID, 2005)

1.6 Estándares RFID

Los estándares de RFID abordan cuatro áreas fundamentales:

1. **Protocolo en la interfaz aérea:** Especifica el modo en el que etiquetas RFID y lectores se comunican mediante radiofrecuencia.
2. **Contenido de los datos:** Especifica el formato y semántica de los datos que se comunican entre etiquetas y lectores.
3. **Certificación:** Pruebas que los productos deben cumplir para garantizar que cumplen los estándares y pueden inter operar con otros dispositivos de distintos fabricantes.
4. **Aplicaciones:** usos de los sistemas RFID.

Como en otras áreas tecnológicas, la estandarización en el campo de RFID se caracteriza por la existencia de varios grupos de especificaciones competidoras. Por una parte está ISO, y por otra *Auto-ID Centre* conocida desde octubre de 2003 como EPCglobal de EPC, (*Electronic Product Code*).
(EstandarEPC, 2006)

1.6.1 Estándares ISO

Los estándares o normalizaciones permiten disponer de soluciones interoperables, que permiten una arquitectura abierta que puede ser implementada por diferentes fabricantes.
Estos son las más relevantes en el entorno RFID:

La ISO (*International Organization for Standaritzacion*) trabaja mediante comités técnicos, que están organizados mediante subcomités formados por grupos de trabajo.

Estándares desarrollados para tarjetas de identificación:

- **ISO/IEC 10536 *Identification cards – Contactless integrated circuit cards*:** para tarjetas de identificación inteligentes a 13,56 MHz. Describe sus características físicas, dimensiones localizaciones de las aéreas de interrogación, las señales electrónicas y los procedimientos de *reset*, las respuestas de *reset* y el protocolo de transmisión.
- **ISO/IEC 14443 *Identification cards – proximity integrated circuit cards:*** desarrollado para tarjetas de identificación inteligentes con rango superior a

un metro, utilizando la frecuencia 13,56 MHz. Describe las características físicas, el interfaz aéreo, la inicialización y anticolisión, y el protocolo de transmisión.

- **ISO/IEC 15693 Contactless integrated circuit cards – Vicinity cards:** se desarrollan las características físicas, la interfaz aérea y los protocolos de transmisión y anticolisión para tarjetas sin contacto con circuitos integrados en la banda HF (13,56 MHz).

Estándares desarrollados para la gestión a nivel unidad:

- **ISO/IEC 15961 RFID *for item management – Data protocol: application inteface:*** dirigido comandos funcionales comunes y características de sintaxis, por ejemplo, tipos de *tags*, formatos de almacenamiento de datos, o compresión de los datos. Los estándares de interfaz aérea no afectan a este estándar.
- **ISO/IEC 15962 RFID *for item management – Protocol: Data encoding rules and logical memory functions:*** dirigido al procedimiento que el sistema RFID utiliza para intercambiar información de la gestión a nivel unidad. Crea un formato de datos uniforme y correcto, una estructura de comandos, el procesamiento de errores.
- **ISO/IEC 15963 *for item management – Unique identification of RF tag*:** este estándar se dirige al sistema de numeración, el proceso de registro y uso del *tag* RFID. Se ha diseñado para el control de calidad durante el proceso de fabricación. También está dirigido a la trazabilidad de los *tags* RFID durante este proceso, su ciclo de vida y control para anticolisión de varios *tags* en la zona de interrogación.
- **ISO/IEC 19762: *Harmonized vocabulary – Part 3: radio-frequency identification:*** documento que proporciona términos generales y definiciones en el área de la identificación automática y técnicas de captura de datos, con secciones especializadas en varios campos técnicos, al igual que términos esenciales para ser usados por usuarios no especializados en comunicaciones. La parte 3 es la que hace referencia a la tecnología RFID.
- **ISO/IEC 18000 Air inteface standards:** diseñada para crear una interoperabilidad global, donde se define la comunicación entre los *tags* y

los lectores. Incluyendo diferentes frecuencias de trabajo. El objetivo del estándar es asegurar un protocolo de interfaz aérea universal. Este estándar contiene 7 partes diferentes. La primera consiste en la arquitectura del sistema RFID para la gestión unitaria. La parte 3 y 6 son las más relevantes y críticas. En la 3 se definen dos modos no interoperables aunque se han diseñado para no interferirse entre ellos. El modo 1 está basado en ISO 15693 y el modo 2 en PJM (modulación) para obtener mayor tasa de bits.

La parte 6 también define dos modos de operación conocidos como A y B.

- **ISO/IEC 18001 RFID *for Item Management - Application Requirements Profiles:*** proporciona el resultado de tres estudios para identificar aplicaciones y usos de la tecnología RFID con gestión a nivel unidad de artículo, con una clasificación resultante según diferentes parámetros operacionales, incluyendo el rango de operación, tamaño de la memoria, etc. (RFIDMagazine, 2011)

1.6.2 Estándares EPCglobal

También EPC global tiene desarrolladas otras especificaciones y estándares:

- **GS1 Códigos de Barra:** El sistema GS1 es un conjunto de estándares que permite la administración eficiente de la cadena de suministro multi-sectorial y mundial, mediante la identificación inequívoca de productos, unidades de embarque, bienes, localizaciones y servicios. Facilita los procesos de comercio electrónico incluyendo el rastreo y seguimiento completos.

  Mediante la aplicación del sistema GS1 es posible obtener significativas mejoras en las operaciones logísticas, una reducción de los costos de los trabajos realizados en papel, una considerable disminución de los tiempos de preparación de órdenes y entregas, así como mayor precisión y una administración más eficiente de toda la cadena de suministro y demanda.
- **GS1 eCom**: provee estándares globales para el intercambio electrónico de mensajes de negocios de forma rápida, eficiente y precisa entre socios comerciales.

  GS1 eCom es un término utilizado para la tecnología de intercambio electrónico de datos, que puede definirse como "la transferencia de datos

estructurados de un computador a otro, por vía electrónica y con un mínimo de intervención humana".

GS1 eCom proporciona dos estándares complementarios para la mensajería de documentos electrónicos.

- **GS1 EANCOM y XML:** El uso de estándares como XMLGS1 y GS1EANCOM® proporciona una estructura estandarizada y previsible de los mensajes de comercio electrónico, permitiendo a los socios de negocios comunicar los datos de negocio con rapidez, eficiencia y precisión, independientemente de su hardware interno o tipos de software.
- **GS1 Trazabilidad:** Estándar Mundial de Trazabilidad. La Trazabilidad es la capacidad de seguir una unidad de producto a lo largo de la cadena de suministros. Son aquellos procedimientos preestablecidos y autosuficientes que permiten conocer el histórico, la ubicación y la trayectoria de un producto o lote de productos a lo largo de la cadena de abastecimiento en un momento dado, a través de unas herramientas determinadas.

  La Trazabilidad consiste en asociar sistemáticamente un flujo de información a un flujo físico de mercancías de manera que se pueda encontrar en un instante determinado la información requerida relativa a los lotes o grupos de productos específicos.
- **GS1 GDSN:** La Red global de Sincronización de Datos (*Global Data Synchronization Network*) es un ambiente global automatizado, basado en estándares que permiten la sincronización segura y continua de datos, permitiendo que todos los socios de negocios, tengan al mismo tiempo información consistente y constante de artículos en sus sistemas.

  La Red Global de Sincronización de Datos (GDSN) conecta a Detallistas y Proveedores, a través de sus *Data Pools* (Catálogos Electrónicos de Datos), con el GS1 *Global Registry*™.

  GDSN es una red basada en Internet, conectada a los Catálogos Electrónicos de Datos (*Data Pools*) y un registro global (el GS1 *Global Registry*™) que permiten a compañías alrededor del mundo intercambiar y sincronizar información logística de productos de forma estándar, a través de la Cadena de Abastecimiento entre sus respectivos socios de negocios.

Como reseña de algunos de estos otros estándares, AIAG (*Automotive Industry Action Group*, que es una asociación con más de 1.600 fabricantes), ha desarrollado junto a EPC global estándares para la industria de la automoción, en específico el "*Application Standard for RFID Devices in the Automotive Industry*" que viene acompañado por otros como "AIAG B - 11", estándar para identificar neumáticos y ruedas con RFID.

Cada región o país tienen normativas técnicas de referencia. Por ejemplo nos podemos encontrar con:

- **EN 300 220 (ETSI):** características técnicas y métodos de medida para equipos de radio de corto alcance funcionando entre 25 y 1.000 MHz, hasta 500 mW de potencia.
- **EN 302 208 (ETSI):** características técnicas y métodos de medida para dispositivos de datos en la banda 865 – 868 MHz, hasta una potencia de 2 W.

También una breve explicación de los temas asociados con los parámetros de distancias, número de *tags* dentro del campo de interrogación, etc. Se incluye una clasificación de los tipos de *tags* según las aplicaciones.

- **EPC™ *Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID:*** creado por EPC global, joint venture entre EAN (*European Article Numbering*) y UCC (*Uniform Code Council*), y tecnología desarrollada por *Auto* – ID *Center*, en este documento se desarrolla el estándar para el protocolo de interfaz aérea de comunicación entre el *tag* y el lector.
- **13.56 MHz *ISM Band Class 1 Radio Frequency* (RF) *Identification Tag Interface Specification*:** desarrollado por EPC global para definir la interfaz de comunicación y el protocolo para la clase 1 en 13,56MHz. Incluye los requerimientos de los *tags* y lectores para establecer comunicaciones en dicha banda de frecuencias.
- ***Application Level Event (ALE) Specification Version* 1.0:** estándar desarrollado por EPC global que especifica un interfaz a través de la cual se filtra y consolida códigos electrónicos EPC con origen de varios dispositivos.

En resumen se observa que los estándares RFID están disponibles para multitud de aplicaciones. La confusión general ha venido por la confrontación entre la ISO 18000 parte 6A o 6B y EPC global que han definido la interfaz aérea para la utilización de

RFID en UHF, aunque EPC global ha definido otros elementos para la utilización del EPC en la cadena de suministro.

Esta confusión o debate puede tener fin próximamente si la ISO tramita lo realizado por EPC global como la parte 6C. De momento coexisten aunque la ISO solo ha definido el protocolo de interfaz aérea, pero no la estructura numérica ni la implementación física del *tag* y los lectores.

(EPCglobalinc, 2009)

1.7 Frecuencias RFID

No hay ninguna corporación pública global que gobierne las frecuencias usadas para RFID. En principio, cada país puede fijar sus propias reglas.

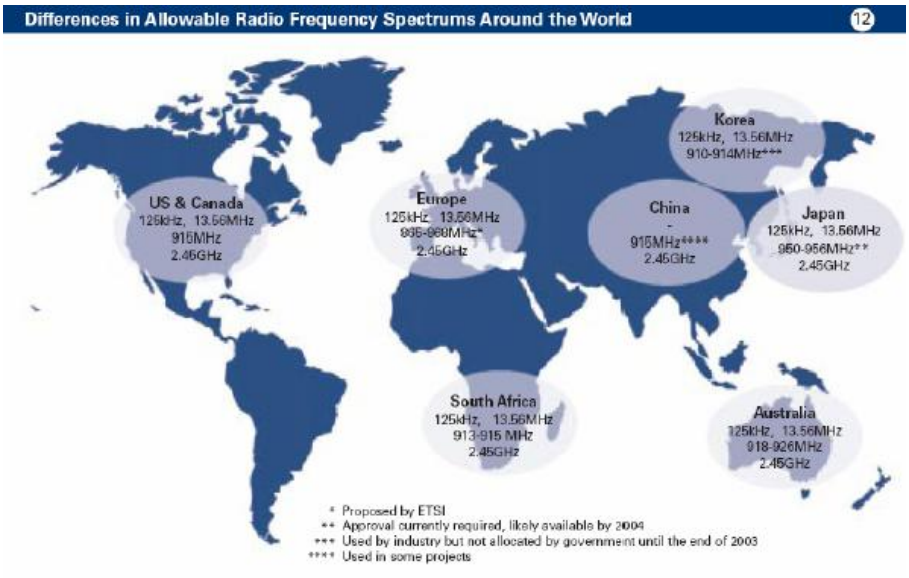


Gráfico 19. Frecuencias utilizadas en cada una de las bandas por los diferentes países.

Los sistemas RFID utilizan diferentes frecuencias, pero generalmente las más utilizadas son:

Frecuencia	Denominación	Rango
125 kHz – 134 kHz	LF (Baja Frecuencia)	Hasta 45 cm.
13,553 MHz – 13,567 MHz	HF (Alta Frecuencia)	De 1 a 3 m.
400 MHz – 1000 MHz	UHF (Ultra Alta Frecuencia)	De 3 a 10 m.
2,45 GHz – 5,4 GHz	Microondas	Más de 10 m.

Gráfico

Gráfico 20. Rangos y Frecuencias RFID

Banda de frecuencias utilizadas en la tecnología RFID

(RFIDSolutions, 2002)

- **LF (*Low Frecuency* o Baja Frecuencia):** su principal ventaja es su aceptación en todo el mundo, funciona cerca de los metales y está ampliamente difundida. La distancia de lectura es inferior a 1,5 metros, por lo que las aplicaciones más habituales son la identificación de animales, barriles de cerveza, *auto key* and *lock* o bibliotecas.

Ventajas:

- No tiene tantas limitaciones en cuanto niveles de potencia permitidos
- Penetra bien en materiales (agua, papel y cartón, tela…)
- Válido para aplicaciones donde, por seguridad y privacidad, se requieren distancias mínimas de lectura

Inconvenientes:

- Rango de lectura muy corto (<<1m)
- Velocidad de lectura lenta
- Maneja pocas cantidades de datos
- Requiere de antenas más grandes

- **HF (*High Frecuency* o Alta Frecuencia):** esta frecuencia también está muy difundida, pero a diferencia de la frecuencia baja, la alta no funciona cerca de los metales. Normalmente se utiliza en aplicaciones tales como la trazabilidad de los productos, movimientos de equipajes de avión o acceso a edificios.

Ventajas:

- Antenas más pequeñas para lecturas a corta distancia
- Penetra bien en materiales (agua, papel, madera, tela) pero no mejor que LF
- Lectores más baratos que los de UHF
- *Tag* más delgado, barato y simple (menos vueltas de la bobina) que para LF
- Mayor tasa de transferencia de datos

Inconvenientes:

- Rango de lectura < 1m
- No penetra metales ni funciona muy bien cerca de ellos
- Requiere de antenas grandes para cubrir mayores distancias (siempre < 1m)

- **UHF (*Ultra High Frecuency* o Ultra Alta Frecuencia):** los equipos que operan a estas frecuencias UHF (*Ultra High Frequency*) no pueden ser

utilizados de forma global porque no existen regulaciones globales para su uso y su aplicación depende de la legalidad del país. Este tipo de frecuencia se usa para aplicaciones de trazabilidad con *tags* activos.

Ventajas:

- Rango de lectura mayor (<10m)
- Antenas aún más pequeñas
- Lectores de menor potencia pero más caros
- Mayor tasa de transferencia de datos
- Posibilidad de uso de antenas direccionales con la ventaja del control de zonas de cobertura
- Mayor capacidad de lectura simultánea
- Posibilidad de utilización de protocolos anti-colisión de acceso al medio

Inconvenientes:

- Menor capacidad de penetración de materiales, sobre todo agua y metales
- Banda de frecuencias y potencias máximas de emisión más reguladas y diferentes en cada país

- **Microondas:** estas frecuencias son las más habituales para los *tags* activos, pero no tienen el problema de la falta de regulaciones globales, además ofrecen largas distancias de lectura y altas velocidades de transmisión. Los *tags* activos que operan en el rango de las microondas son muy usados para seguimiento y trazabilidad de personas u objetos.

Ventajas:

- Tamaño del *tag* aún más pequeño
- Mayor rango de lectura (<100m)
- Mayor ancho de banda que en UHF
- Antenas más pequeñas
- La mayor tasa de transferencia de datos
- Menor capacidad de penetración de materiales absorbentes
- Zonas de cobertura aún más definidas
- Con ciertas modificaciones de diseño se podrían utilizar en entornos cercanos a metales

Inconvenientes:

- Susceptible al ruido electromagnético

- Espectro de frecuencias más ocupado: redes inalámbricas WiFi
- Generalmente sólo se utiliza esta banda con *tags* activos
- Las regulaciones de frecuencias y potencias todavía están en fase de regulación en muchos países

(AITEX, 2006)

1.8 Beneficios y Ventajas de la tecnología RFID

- La confiabilidad en una solución de RFID es extremadamente alta y tiene la menor tasa de error de todas las tecnologías incluyendo códigos de barras, cintas magnéticas y equipos biométricos.
- Las *tags* no necesitan contacto visual con el lector.
- Incrementan la eficiencia y productividad de la empresa.
- Automatiza eliminando el error humano la identificación, conteo, rastreo, clasificación y ruteo de cualquier objeto.
- Mejora la recolección de datos e identificación de todo tipo de productos.
- Elimina errores y por consecuencia el desperdicio de recursos.
- Mejora la administración de información.
- Mejora la administración y el manejo de materiales.
- Reduce costos operativos y de producción.
- Reduce inventarios.
- Incrementa el control de calidad.
- Mejora la calidad del servicio al cliente.
- Incrementa la información para el departamento administrativo y el cliente.

1.9 Desventajas de la tecnología RFID

- La instalación de este sistema en seres humanos violaría el derecho a la privacidad y a la intimidad.
- La utilización de este sistema en las diferentes áreas de trabajo, produciría un alto grado de desempleo.
- El costo de los productos se elevaría.

En cuanto a la privacidad

- El uso de la tecnología RFID ha causado una considerable polémica de productos. Las cuatro razones principales por las que RFID resulta preocupante en lo que a privacidad se refiere son:
- La etiqueta puede ser leída a cierta distancia sin conocimiento por parte del individuo.
- Si un artículo etiquetado es pagado mediante tarjeta de crédito o conjuntamente con el uso de una tarjeta de fidelidad, entonces sería posible enlazar la ID única de ese artículo con la identidad del comprador.
- El sistema de etiquetas EPCGlobal crea, o pretende crear, números de serie globales únicos para todos los productos, aunque esto cree problemas de privacidad y sea totalmente innecesario en la mayoría de las aplicaciones.

Capítulo II. Manual de clases de la Tecnología Alien

2.1 Introducción

El siguiente manual proporciona instrucciones básicas para controlar mediante programación un Lector o *Reader*, usando el lenguaje de programación Java y las clases más importantes suministradas por la Tecnología Alien, como una herramienta de apoyo para desarrolladores de software y usuarios.

El *Reader* de RFID Alien puede controlarse usando un número de sistemas y lenguajes. Este manual se centra en controlar el *Reader* usando la biblioteca de Java suministrada por Alien.

2.2 Clases de Alien

Se dividió a la biblioteca en cinco grupos funcionales (paquetes de Java) para el control de varios aspectos del *Reader*:

1. ***Reader*** - clases para la comunicación con un Lector o *Reader*
2. ***Tags*** - clases relacionadas con *Tags* o *tags* RFID y datos EPC
3. ***Discovery*** - clases para conocer las ubicaciones de los *Reader* conectados por RS-232 o Ethernet
4. ***Notify*** - clases para escuchar notificaciones "*push*" y transmitir datos de los *Reader* través de la LAN
5. ***Util***- clases para realizar operaciones de bits, convertir entre cadenas hex/ASCII/binario, analizadores XML, manejo de puerto serie y para determinar la versión de la API.

2.2.1 *Reader Classes*

2.2.1.1 Introducción

ReaderClasses son lo principal para la comunicación con un *Reader* en la red o un puerto serial. Normalmente, el objeto *Reader* se obtiene de un objeto *DiscoveryItem*. Sin embargo, si se conoce la ubicación (puerto serial o dirección de red) del *Reader*, un objeto *Reader* puede crear instancias directamente sin necesidad de *Discovery Classes*.

La clase padre, *AbstractReader*, proporciona la funcionalidad básica de una clase *Reader* genérica, el manejo serial, las comunicaciones de red, y métodos de bajo nivel para el envío de comandos a un *Reader* y la recepción de respuestas. Como su nombre indica es una clase abstracta y por lo tanto no puede ser instanciada, por lo que una de estas subclases es usada en su lugar:

- ***AlienClass1Reader*** es la clase que representa a todos los *Reader* pasivos de Alien Class I, y expone todo de los comandos *Reader* de los *Reader* pasivos a través de su API. La mayoría de los lectores Alien son manejados mediante esta clase.
- ***AlienClassOEMReader*** se extiende de *AlienClass1Reader* para comunicarse con módulos OEM RFID de Alien (*Reader* de lectura/escritura), usando un protocolo binario. La comunicación serial solo es posible con Módulos OEM.
- ***AlienClassBPTReader*** se extiende de *AlienClass1Reader* e incluye toda la funcionalidad adicional del *Reader* de *tags* que funcionan con baterías o *Battery Powered Tag*.

2.2.1.2. Creación de un *ReaderObject* desde *DiscoveryItem*

Si estamos usando *Discovery Classes*, cualquiera de los *Reader* que se encuentran en la red o en puertos seriales están representados por objetos *DiscoveryItem*. Para derivar un objeto *Reader* desde *DiscoveryItem*, se utiliza el siguiente método:

AlienClass1ReaderReader = *DiscoveryItem*.get*Reader* ();

Si el *Reader* descubre una *Battery Powered Tag* o un Módulo OEM, el objeto *Reader* que devuelve será de una clase apropiada, y puede ser:

```
AlienClass1Reader class1Reader = discoveryItem.getReader();

if (class1Reader instanceof AlienClassBPTReader) {
    AlienClassBPTReader classBPTReader = (AlienClassBPTReader)class1Reader;
} else if (class1Reader instanceof AlienClassOEMReader) {
    AlienClassOEMReader classOEMReader = (AlienClassOEMReader)class1Reader;
}
```

Gráfico 21. Código para la composición de un objeto *Reader*

2.2.1.3. Instanciar Directamente un *Reader*

Si se sabe que existe un *Reader* en un puerto serial o una dirección de red, un nuevo objeto *Reader* puede ser creado directamente sin tener que utilizar las clases Discovery. Una manera de hacer esto es instanciar el objeto *Reader* usando constructores sin parámetros, y luego usar el método *setConnection*() para especificar el nombre del puerto serial o la dirección de red al que está conectado.

Ejemplo conexión de *Reader* con Puerto Serial:

AlienClass1ReaderReader = newAlienClass1Reader();
Reader.*setConnection*("COM1");

Ejemplo conexión de *Reader* con una Dirección de Red:

AlienClass1ReaderReader = new AlienClass1Reader();
Reader.*setConnection*("1.2.3.4", 23);
Reader.*setUsername*("alien");
Reader.*setPassword*("password");

La dirección IP del *Reader* es un *String* y el número de puerto que se utiliza para comandos de red es un *Integer*, normalmente se utiliza el puerto 23.
Antes de conectarse a un *Reader* por medio de una red, se debe especificar el Nombre de Usuario y contraseña usando los métodos *setUsername()* y *setPassword()*. Generalmente todos los *Reader* usan "alien" como *Username* y "password" en *Password*.

La *AlienClass1Readerclass* autoriza automáticamente cuando se abre una conexión desde el socket, pero se debe especificar primero los valores.

Hay otros métodos utilizados para especificar el puerto serie o dirección de red:

AlienClass1Reader.*setSerialConnection*("COM1");
AlienClass1Reader.*setNetworkConnection*("1.2.3.4", 23);
AlienClass1Reader.*setNetworkConnection*("1.2.3.4:23");

También puede especificar esta información en el constructor:

AlienClass1ReaderReader = new *AlienClass1Reader*("COM1");
AlienClass1ReaderReader = new *AlienClass1Reader*("1.2.3.4:23");

2.2.1.4. Apertura y Cierre de una conexión *Reader*

Una vez que se ha instanciado el objeto *Reader* y configurado los parámetros de conexión, se puede enviar los comandos y pedir los datos.

Para abrir una conexión de *Reader,* se utiliza el siguiente método:

Reader.open();

Si la conexión falla, se produce un Alien*ReaderConnectionException*.

Se puede realizar una prueba en el objeto *Reader* para ver si la conexión está abierta con:

Reader.isOpen();

Este método devuelve true si la conexión está abierta, o false si no.

Finalmente, se usa el siguiente método para cerrar la conexión:

Reader.close();

2.2.1.5. Comunicación con un *Reader*

Todos los comandos desde y hacia el *Reader* son mensajes de texto basados en ASCII en donde forman pares de respuestas en comandos (Modulo OEM usa un protocolo de comunicación binario, el cual se oculta desde *AlienClassOEMReader class*). La manera más básica para comunicarse con un *Reader* es usando un método llamado *doReaderCommand()*, el cual envía un comando en ASCII y responde en ASCII.

*StringReader*Name = *Reader.doReaderCommand*("*get ReaderName*");

Sin embargo, este método requiere el conocimiento de un conjunto de comandos de lectura para analizar y procesar las respuestas de los *Reader*.

Las clases de lectura proveen muchos métodos adicionales que corresponden directamente con el conjunto de comandos *Reader*.

Por ejemplo, existe un método llamado *getReaderName*(), el cual devuelve el nombre del *Reader*. De igual manera, el método llamado *getPersistTime*(), es igual que hacer *doReaderCommand*(“*get PersistTime*”), y luego analizar la respuesta en *string*, encerrada dentro de un valor entero.

2.2.1.6. *AlienClass1Reader* Class

La *AlienClass1Reader* class proporciona el acceso a los comandos base de todas las clases *Alien Class* 1 RFID. En lugar de manejar clases individuales por cada modelo de lectura, esta clase común las maneja a todas.

Cada uno de los atributos del *Reader* son expuestos por los métodos *getter* y *setter*, así como otros comandos como *notifyNow(), autoModeReset(), programTag(),* etc.

2.2.1.7. *AlienClassBPTReader* Class

Alien Battery Powered TagReaders soportan comandos extras especialmente diseñados a tomar ventaja del mejoramiento de la funcionalidad de Battery *Tags*. Estos comandos adicionales se dividen en tres categorías:

1. ***Memory* o Memoria**

Las *Tags* con batería o *BatteryTags* pueden soportar opcionalmente la lectura-escritura en su memoria, normalmente en los rangos de 4K a 16K bytes. Los comandos de memoria permiten leer y escribir en la memoria de la etiqueta en bloques vía RF.

2. ***Sensors* o Sensores**

Las *BatteryTags* pueden soportar el uso de sensores como la temperatura o la vibración. Los comandos del sensor pueden ser usados para preguntar y controlar el uso de estos dispositivos.

3. ***Logging* o Inicio de Sesión**

Si una etiqueta está equipada con uno o más sensores y una memoria, puede registrar datos de manera autónoma para etiquetar la memoria incluso en ausencia de un campo RF. Los comandos *logging* están en la interfaz para su funcionalidad.

2.2.1.7.1 *Battery Powered Tag Public Methods*

Presentamos una breve lista de los métodos de la clase *AlienClassBPTReader*:

- ***public Tag getTagID(String tagID)***

  Devuelve el ID de una etiqueta única especificada por los comandos de la máscara.

- ***public String getTagInfo(String tagID)***

  Devuelve información acerca de una sola etiqueta.

- ***public int getSensorValue(String tagID)***
  Devuelve el valor del sensor de la etiqueta

- ***public boolean isLogging(String tagID)***
  Devuelve el estado del modo de registro de una etiqueta específica

- ***public void setLogging(String tagID, boolean isLogging)***
  Habilita o deshabilita el logging de una etiqueta específica.

- ***public byte[] getLoggingInterval(String tagID)***
  Devuelve la periodicidad en la que la etiqueta registra los datos del sensor en la memoria de la etiqueta.

- ***public void setLoggingInterval(String tagID, int hours, int mins, int secs)***
  Fija la periodicidad en la que la etiqueta registra los datos del sensor a la memoria de la etiqueta.

- ***public short[] getMemory(String tagID, int lengthIndex, int startIndex)***
  Devuelve un tramo de la memoria de la etiqueta de una etiqueta especificada.

- ***public boolean setMemory(String tagID, int startIndex, byte byteArray[])***
  Almacena una serie de bytes dentro de la memoria de la etiqueta.

- ***public void clearMemory(String tagID)***
  Borra completamente la memoria de una etiqueta específica.

- ***public int getMemoryPacketSize()***
  Devuelve el número de bytes a utilizar en cada paquete de memoria hacia y desde la etiqueta.

- ***public void setMemoryPacketSize(int memoryPacketSize)***
  Especifica el número de bytes a utilizar en cada paquete de memoria hacia y desde la etiqueta.

2.2.1.8. *AlienClassOEMReader class*

Aunque AlienClassOEM*Reader* hereda de *AlienClass1Reader*, su implementación es muy diferente. Esto es porque mientras todos los demás *Reader* RFID se comunican con un comando de respuesta ASCII, el Módulo OEM usa un protocolo binario. La diferencia es clara comparando incluso un comando simple como "observa las *tags* y dime que ves".

ASCII-Based Protocol Example	
Command	get Taglist
Response	0102 0304 0506 0708 8000 8004 13DB 34DE
Binary Protocol Example	
Command	10 01 21 00 40 00 01 03 03 00 00 01 04 EE 10 02
Response	21 49 40 01 D1 3B 21 49 40 02 00 00 08 01 02 03 04 05 06 07 08 4C D8 21 49 40 02 00 00 08 80 00 80 04 13 DB 34 DE 4C D8 21 49 40 03 00 02 D2 F4 D3 95 CC F4

Gráfico 22. Protocolo ASCII / Protocolo Binario

AlienClassOEM*Reader*, reemplaza los métodos de envío y recepción de *AlienClass1Reader* tomando cuidado de empaquetar y desempaquetar los comandos de respuesta de los *Reader*. Este también reemplaza los métodos de los comandos del *Reader* que se aplican al módulo OEM, sustituyendo las simples cadenas de comandos ASCII con sus homólogos en el protocolo binario.

El conjunto de comandos del módulo OEM es un subconjunto de otros *Reader* Alien, y algunas operaciones como *AutoMode* y *NotifyMode* no se encuentran. Para compensar esto, algunos de las funciones de alto nivel toman a AlienClassOEM*Reader*. Por ejemplo, la clase implementa una versión muy básica de AutoMode, convirtiéndola en la causa de porque la clase pide al *Reader* leer las *tags*, guardando las *tags* detectadas en una *tag*list interna de la clase.

AlienClassOEM*Reader* también admite una interfaz de línea de comandos básica, ésta herramienta permite hablar a un módulo OEM utilizando el mismo (aunque más restringido) protocolo ASCII:

```
Alien>help
*********************************************
*
* Help
*
*********************************************
GENERAL:
    Help (H)
    Info (I)
    ! (Repeat Last Command)
    Get ReaderName
    Get ReaderType
    Get ReaderVersion
    Get MfgInfo
    Get|Set AntennaSequence (i, j, k...)
    Get|Set Antenna
    Get ExternalInput
    Set ExternalOutput
TAGLIST:
    Get|Set AcquireMode
    Get|Set AcqCycles
    Get|Set AcqCount
    Get|Set AcqEnterWakeCount
    Get|Set AcqExitWakeCount
    Get|Set AcqSleepCount
    Get|Set PersistTime (secs)
    Get TagList [n]
    Clear TagList
    Wake
    Sleep
    Get|Set Mask (All | bitLen, bitPtr, XX XX)
AUTONOMOUS MODE:
    Get|Set Automode (On or Off)
    AutoModeReset
PROGRAMMING:
    Program Tag = XX XX XX XX XX XX XX XX
    Verify Tag
    Erase Tag
    Kill Tag = XX XX XX XX XX XX XX XX YY
    Lock Tag = YY
MISC:
    Get Timer
    Set ReaderCommand = XX XX XX...
    RC = XX XX XX...

(XX = TagID byte)
(YY = LockCode byte)

Alien>get taglist
Tag=0102 0304 0506 0708  Disc=Fri Jun 18 12:58:18 PDT 2004  Last=Fri Jun 18
12:58:18 PDT 2004  Ant=0  Count=8

Alien>set AntennaSequence = 0,1
AntennaSequence (i, j, k...) = 0, 1
```

Gráfico 23. Ejemplo de la interfaz de aplicación con OEM Module

Esto no quiere decir que puede conectar un host a un módulo OEM y comunicarse con el software de terminal, utilizando el protocolo de comandos ASCII, pero la capacidad existe para que la clase AlienClassOEM*Reader* pueda reaccionar a los comandos basados en texto. Esto se realiza por reemplazar los métodos *sendString()* y *receiveString()* de Abstract*Reader*, captura de cadenas comandos conocidos y manejarlos con otros métodos que son privados para la clase.

2.2.1.8.1. *AlienDLEObject Class*

El protocolo binario utilizado para la comunicación con el modulo OEM no es tan sencillo como el protocolo ASCII. Los comandos, parámetros y valores devueltos están especificados como secuencias de bytes, y se empacan para su transmisión

hacia y desde el Modulo OEM. Este paquete comprende la adición de los bytes *ReaderNumber* y *SessionID*, DLE(*Data Link Escape*) y SOM(*Start of Message*) al inicio del mensaje, y EOM(*End of Message*) y DLE al final del mensaje. Por otra parte, los bytes CRC deben ser calculados y añadidos al paquete para verificar la transmisión.

Estas tareas se hacen más fáciles con la clase *AlienDLEObject*, que proporciona constantes para todos los comandos, subcomandos y los códigos de respuesta que el módulo OEM comprende. Las series *prepareGenericCommand()* de los métodos le permiten simplificar los parámetros, comandos y el empaque del comando.

```
AlienClassOEMReader reader = new AlienClassOEMReader("COM1");

AlienDLEObject command = new AlienDLEObject();

// Turn all external digital outputs off
command.prepareGenericCommand(readerCommand.CMD_SET_IO_PORT_VALUE, 0);

reader.issueReaderCommand(command);
```

Gráfico 24. Ejemplo creación del Comando DLE

La respuesta del *Reader* está cuando el método *issueReaderCommand()* de *AlienClassOEMReader*realiza el envío del comando, espera a que el *Reader* responda y llena un número de campos con los datos en el objeto AlienDLE*Object*.

- ***public byte[] replyBuffer;***

  Es donde se almacenan los bytes sin formato de la respuesta del *Reader*.

- ***public int replyCommType;***

  CommType es el código de respuesta del *Reader*, que indica éxito (0 x 00), condición no error (< 0 x 80), o una condición de error (> 0 x 80).

- ***public String replyCommTypeMessage;***

  Devuelve una cadena legible describiendo el CommType.

- ***public int replyValueInt;***

  Si los datos de respuesta están en un solo byte (por ejemplo como, *Reader*Number), este va a ser un int y se almacenaran aquí para acceder con más rapidez.

- ***public byte[] replyValueHexArray;***

  Aquí se almacenan todos los bytes de datos de respuesta para un acceso rápido.

- ***public int[] replyValueIntArray;***

  Todos los bytes de datos de respuesta son convertidos en enteros y almacenados aquí para acceder con más rapidez.

2.2.1.9. Alien *Reader Class Exceptions*

Las clases en el paquete *com.alien.enterpriseRFID.Reader*manejan los errores lanzando excepciones hacia el objeto de llamada. Hay seis excepciones *Reader*, todos se extienden desde una sola clase *AlienReaderException*:

- *AlienReaderCommandErrorException*
- *AlienReaderConnectionException*
- *AlienReaderInvalidArgumentException*
- *AlienReaderNoTagException*
- *AlienReaderNotValidException*
- *AlienReaderTimeoutException*

2.2.1.9.1. AlienReaderCommandErrorException

Esta excepción se puede producir si el *Reader* responde a un comando con un error.

2.2.1.9.2. AlienReaderConnectionException

Esta excepción se produce, si hay un problema al conectar a un *Reader*. Por ejemplo, si se intenta una conexión serial pero el puerto serial ya está en uso por otra aplicación, o se trata una conexión de red pero el *Reader* no se encuentra en la red.

2.2.1.9.3. AlienReaderInvalidArgumentException

Esta excepción puede ser lanzada por la clase AlienClassOEM*Reader*, si se envía un comando al *Reader* con parámetros fuera de rango o un valor de parámetro incorrecto.

2.2.1.9.4. AlienReaderNoTagException

Esta excepción puede ser lanzada por *AlienClassBPTReader* si se envía un comando a una Battery *Tag* específica, pero la etiqueta no puede ser encontrada.

2.2.1.9.5. AlienReaderNotValidException

Esta excepción puede ser lanzada por cualquiera de las clases *Reader* si se intenta una conexión pero el dispositivo en el otro extremo no es un *Reader* RFID de Alien.

2.2.1.9.6. AlienReaderTimeoutException

Esta excepción puede ser lanzada por cualquiera de la clases *Reader* si cualquier transacción *Reader* toma más tiempo que la duración del tiempo de espera. El tiempo por default es 10 segundos, pero puede ser cambiado con el método *setTimeOutMilliseconds().*

2.2.2. Tag Classes

2.2.2.1. Introducción

Las *tags* desempeñan un papel muy importante en el sistema de *tags* y *Reader* RFID. Por esta razón hay una clase dedicada a almacenar y manipular información de las *tags*. Las clases adicionales y una interfaz son útiles para la gestión de los datos de *tags* y listas de *tags* dentro de sus propias aplicaciones.

2.2.2.2. Reading Tags

Las clases *Reader* permiten a las *tags* para ser leídas por el *Reader* y devolver los resultados como un objeto único de etiqueta o una matriz de objetos de la etiqueta:

Tag[ ] tagList = Reader.getTagList();

También se puede obtener datos de la etiqueta desde un *Reader* al recibir un mensaje de notificación de la misma. En cualquier caso, cada etiqueta devuelta está representada por un objeto *Tag*, que encapsula no sólo la etiqueta ID sino también los datos que le indiquen de dónde, cuándo y cuántas veces fue leída la etiqueta.

2.2.2.3. Tag class

La clase *Tag* se compone de los siguientes miembros, cada uno es accesible a través de captadores y definidores en la API:

- ***Tag ID* -** Representa el código EPC de la etiqueta.
- ***Discover Time***– El tiempo en que la etiqueta fue vista por primera vez por el *Reader*.
- ***Last Seen Time***– El tiempo en que la etiqueta fue vista por última vez por el *Reader*.
- ***Count***– El número de veces que el *Reader* ha leído la etiqueta desde que esta fue vista por primera vez.
- ***Antenna***– El número de trasmisión de la antena, donde la etiqueta fue vista la última vez.
- ***Protocol***– El protocolo usado para adquirir los ID de las *tags*.
- ***TransmitAntenna & ReceiveAntenna***– Sistema de antena Multi-Estática, como el ALR-9800 puede indicar cuál de las antenas está transmitiendo y cuál de las antenas recibía cuando la etiqueta fue adquirida.
- ***G2Data***– Opcionalmente, el *Reader* puede recuperar datos adicionales de la etiqueta, además del EPC.
- ***RSSI (Return Signal Strength Indication)***– El *Reader* puede dar un informe sobre la indicación de la fuerza de señal de la etiqueta.
- ***Speed & SmoothSpeed***– El *Reader* puede medir la velocidad de la etiqueta. Las mediciones repetidas pueden generar un valor de velocidad suavizada.
- ***SmoothPosition***– Las mediciones de velocidad repetidas integradas con el tiempo nos da una medida de la posición de la etiqueta.
- ***Direction***– Si la etiqueta se aproxima o se aleja de la antena (requiere velocidad de los datos).

2.2.2.3.1. Tag Public Methods

Algunos de los métodos public más útiles proporcionados por un *Tag* se enumeran a continuación:

- ***public String getTagID()***

  Devuelve el ID de la etiqueta.

- ***public int getAntenna()***

Devuelve la antena que fue vista por la etiqueta por última vez.

- ***public long getDiscoverTime()***

Devuelve el tiempo que la etiqueta fue vista por primera vez por el *Reader*.

- ***public long getRenewTime()***

Devuelve el tiempo que la etiqueta fue vista por última vez por el *Reader*.

- ***public int getRenewCount()***

Devuelve el número de veces que la etiqueta ha sido leída.

- ***public int getRSSI()***

Devuelve la última medición de RSSI de la etiqueta.

- ***public int getSmoothSpeed()***

Devuelve el promedio de velocidad de esta etiqueta.

- ***public int getSmoothPosition()***

Devuelve la posición de la etiqueta, en relación a su primera lectura.

- ***public int getDirection()***

Devuelve la dirección aparente de la etiqueta, cerca o lejos de la antena.

2.2.2.4. TagUtil Class

Métodos estáticos para el análisis y decodificación de los datos que están disponibles en *Tag*Util:

- ***public static Tag[] decodeTagList(String tagList)***

Decodifica un mensaje de la lista de etiqueta basada en texto en una matriz de *tags*.

- ***public static Tag decodeTag(String tagData)***

Decodifica una línea de texto de los datos de la lista de *tags* en un elemento de etiqueta.

- ***public static Tag[] decodeXMLTagList(String xmlTagList)***

Decodifica la información de una etiqueta individual de un mensaje de la etiqueta basada en XML.

- ***public static Tag decodeXMLTag(String xmlTagData)***

Decodifica la información de una etiqueta individual de un mensaje de *tags* basado en XML.

- ***public static Tag[] decodeCustomTagList (String tagList, String customFormatString)***

Decodifica un mensaje de la lista de *tags* con formato personalizado en una matriz de *tags*, mediante la definición de *TagListCustomFormat*.

- ***public static Tag decodeCustomTag (String tagLine, String customFormatString)***

Decodifica una línea de datos de la lista de *tags* en un solo elemento de la etiqueta, mediante la definición de *TagListCustomFormat* suministrada.

- ***public static void setCustomFormatString(String customFormatString)***

Precarga la clase *TagUtil* con cadena de *TaglistCustomFormat* actual del *Reader*. Utiliza esto para posteriormente decodificar datos de la lista de *tags* con formato personalizado.

- ***public static Tag[] decodeCustomTagList(String tagLine)***

Decodifica un mensaje de la lista de *tags* con formato personalizado en una matriz de *tags*, utilizando la última *TagListCustomFormat.*

2.2.2.5. TagTable Class

TagTable mantiene un HashTable de las *tags*, con métodos para agregar y quitar *tags*, como ganchos para la notificación de su aplicación sobre los cambios a la lista:

- ***public boolean addTag(Tagtag)***

Agrega una etiqueta a este *TagTable.*

- ***public boolean removeTag(Tagtag)***

Quita una etiqueta de este *TagTable.*

- ***public boolean removeOldTags()***

Elimina *Tags* de este *TagTable* cuyo TimeToLive ha llegado a cero.

- ***public int getPersistTime()***

Devuelve el tiempo de persistencia para *tags* en este *TagTable.*

- ***public void setPersistTime(int persistTime)***

Especifica el tiempo de persistencia para *tags* en este *TagTable*.

- ***public Tag[] getTagList()***

Devuelve la lista de *tags* en este *TagTable*, como una matriz de objetos de la etiqueta.

- ***public TagTableListener getTagTableListener()***

  Devuelve el objeto que ha sido registrado con este *TagTable* para recibir eventos cuando cambia la lista.

- ***public void setTagTableListener(TagTableListener TagTableListener)***

  Registra una *TagTable*Listener con este *TagTable* para ser notificado cuando la lista cambia.

2.2.2.6. TagTableListener Interface

La interfaz usada para comunicar los cambios de *TagTable* a otros objetos. Requiere tres métodos para ser implementado:

1. *public void tagAdded(Tagtag);*
2. *public void tagRenewed(Tagtag);*
3. *public void tagRemoved(Tagtag);*

Un objeto que implementa esta interfaz puede ser registrado con un *TagTable* y estos métodos podrían ser llamados cada vez que las *tags* son agregadas, actualizadas o eliminadas de la *TagTable*.

2.2.3. *Discovery Class*

2.2.3.1. Introducción

Para utilizar y controlar un *Reader*, es preciso conocer primero su dirección de red o el número de puerto serial en el host donde está conectado. La biblioteca de AlienRFID.jar proporciona las clases, *SerialDiscoveryListenerService* y *NetworkDiscoveryListenerService*, es necesario buscar automáticamente a los *Reader* con estos modos de conexión.

En ambos casos, se crea el servicio, un objeto está registrado como receptor de eventos de detección, y se inicia el servicio. Una vez iniciado el servicio se ejecuta en su propio subproceso hasta que se detiene automáticamente (para el descubrimiento serial) o se le dice que se detenga (para la detección de la red).

2.2.3.2. DiscoveryListener Interface

Para que un objeto pueda recibir eventos discovery de una de las clases de servicios *discovery*, este debe implementar la interfaz *DiscoveryListener*. Esta interfaz se define de la siguiente manera:

```
public interface DiscoveryListener {
    public void readerAdded  (DiscoveryItem discoveryItem);
    public void readerRenewed(DiscoveryItem discoveryItem);
    public void readerRemoved(DiscoveryItem discoveryItem);
}
```

Gráfico 25. Definición de la Interfaz de DiscoveryListener

Cuando se ejecuta un servicio *discovery* y descubre la primera vez al *Reader*, éste llama al método *ReaderAdded*() de los oyentes registrados. El conocimiento de este *Reader* es mantenido por el servicio, y si el mismo *Reader* es descubierto nuevamente, los servicios llaman al método *Reader*Renewed() del oyente. Si el servicio *discovery* pierde la pista de un *Reader*, el método *ReaderRemoved*() del oyente es llamado.

Cada método es manejado con un parámetro, un *DiscoveryItem*. Para recuperar una matriz de *Reader* conocidos, se llama al método get*DiscoveryItem*s del servicio *discovery*.

2.2.3.3. DiscoveryItem Class

Esta clase contiene puntos de información clave que permiten a cualquier sistema identificar y ponerse en contacto con el *Reader* descubierto, información como *Reader*Name, *Reader*Type y la dirección. *DiscoveryItem* son proporcionados a través de una serie de captadores y definidores, pero existe un método para traducir esta información en una clase *Reader*:

publicAlienClass1Reader getReader() throws Exception

La llamada a este método *DiscoveryItem* devuelve un objeto *AlienClass1Reader*, el cual se utiliza para interconectar directamente con un *Reader*.

2.2.3.3.1. DiscoveryItem Public Methods

Algunos de los métodos public más útiles proporcionados por *DiscoveryItem*, se enumeran a continuación:

- ***public String getReaderName()***

  Devuelve el nombre del *Reader* descubierto.

- ***public String getReaderType()***

  Devuelve el tipo de *Reader* descubierto.

- ***Public String getReaderAddress()***

  Devuelve la dirección del *Reader* descubierto

- ***public String getReaderMACAddress()***

  Devuelve la dirección MAC del *Reader* descubierto, si el *Reader* lo facilita (caso contrario null).

- ***public String getConnection()***

  Devuelve el método de la conexión del *Reader* "serial" o "network"

- ***public int getCommandPort()***

  Devuelve el número de puerto que usa el *Reader* descubierto para aceptar comandos en la red.

- ***public int getLeaseTime()***

  Devuelve la cantidad de tiempo hasta que el *Reader* descubierto obedece que se debe enviar otro mensaje.

- ***public long getFirstHeartbeat()***

  Devuelve el momento en que *DiscoveryItem* registró por primera vez una señal de su *Reader*.

- ***public long getLastHeartbeat()***

  Devuelve el momento en que *DiscoveryItem* registró por última vez una señal de su *Reader*.

- ***public String getReaderVersion()***

  Devuelve la cadena *Reader*Version del *Reader* descubierto.

- ***public AlienClass1Reader getReader()***

  Este método crea un objeto *AlienClass1Reader* desde el *DiscoveryItem*.

2.2.3.4. SerialDiscoveryListenerService Class

Discovery de un *Reader* conectado al puerto serial de un host, es simplemente revisar cada puerto serial por la presencia de un *Reader* Alien. Esto se logra usando la clase *SerialDiscoveryListenerService*, como en el siguiente ejemplo:

```
import com.alien.enterpriseRFID.discovery.*;

public class SerialDiscoveryTest implements DiscoveryListener {

public SerialDiscoveryTest() {
    SerialDiscoveryListenerService service = new SerialDiscoveryListenerService();
    service.setDiscoveryListener(this);
    service.startService();
    ...(application continues)...
}

public void readerAdded(DiscoveryItem discoveryItem) {
    System.out.println("Added:\n" + discoveryItem.toString());
}

public void readerRenewed(DiscoveryItem discoveryItem) {
    System.out.println("Renew:\n" + discoveryItem.toString());
}

public void readerRemoved(DiscoveryItem discoveryItem) {
    System.out.println("Removed:\n" + discoveryItem.toString());
}
```

Gráfico 26. Ejemplo de SerialDiscovery

Cuando el objeto *SerialDiscoveryListenerService* está instanciada e iniciada, adquiere automáticamente una lista de todos los puertos serial en el host y luego procede a interrogar a cada puerto, en busca de un *Reader*. Si se encuentra un *Reader*, o en posteriores análisis un *Reader* se pierde o se renueva, se llama al método adecuado de *DiscoveryListener*.

Además, un *ActionListener* puede ser registrado con *SerialDiscoveryListenerService*. El método *actionPerformed()* es llamado cuando el *SerialDiscoveryListenerService* explora cada puerto, y puede ser utilizado como un seguimiento.

2.2.3.4.1. SerialDiscoveryListenerService Public Methods

Algunos de los métodos más útiles son proporcionados por *DiscoveryItem* se listan a continuación, los Métodos *public NetworkDiscoveryListenerService* son los mismos que *SerialDiscoveryListenerService*.

- ***public void setDiscoveryListener(DiscoveryListener discoveryListener)***

Registra un objeto con servicios discovery para recibir mensajes cuando los *Reader* son descubiertos, renovados o eliminados.

- ***public void startService()***

Inicia el servicio discovery.

- ***public void stopService()***

Detiene o pausa el servicio discovery

- ***public boolean isRunning()***

Devuelve verdadero si el servicio discovery se encuentra corriento todavia.

- ***public DiscoveryItem[] getDiscoveryItems()***

Devuelve una matriz de *DiscoveryItem*s en representación de todos los *Reader* que el servicio discovery conoce.

- ***public void setMaxSerialPort(int maxPort)***

Establecer el número máximo de puertos COM para buscar.

- ***public void setSerialPortList(String portList)***

Especifica una lista separada por comas de los números de puertos COM a escanear.

2.2.3.5. NetworkDiscoveryListenerService Class

Cada *Reader* Alien está configurado de forma predeterminada, para transmitir mensajes sobre su subred local. Estos mensajes son de UDP (*User Datagram Protocol*) paquetes que contienen documentos pequeños XML que detallan el tipo de *Reader*, nombre e información de contacto. Al escuchar estos mensajes, las *networkDiscovery Class* pueden identificar e informar los detalles de los *Reader* que existen en la red.

La clase que realiza estas tareas de escucha se llama *NetworkDiscoveryListenerService*. Una vez que esta clase es instanciada e iniciada, se ejecutará en su propio subproceso hasta que se detenga. Mientras se está ejecutando, escucha los latidos de los *Reader* en el puerto de escucha (por defecto es 3988), o bien llamando a los métodos *Reader*Added () o *Reader*Renewed () de un *DiscoveryListener* registrado cuando detecta un *Reader*. Parte del latido enviado por el *Reader* indica el tiempo hasta que se espera el siguiente latido. Pasado este plazo

antes de que el siguiente latido sea recibido, el servicio asume que el *Reader* se ha desconectado y llamará al método *ReaderRemoved().*

El siguiente código demuestra cómo realizar el descubrimiento de la red:

```
import com.alien.enterpriseRFID.discovery.*;

public class NetworkDiscoveryTest implements DiscoveryListener {

public NetworkDiscoveryTest() throws Exception {
    NetworkDiscoveryListenerService service = new
NetworkDiscoveryListenerService();
    service.setDiscoveryListener(this);
    service.startService();
    ...(application continues)...
}


public void readerAdded(DiscoveryItem discoveryItem) {
    System.out.println("Added:\n" + discoveryItem.toString());
}

public void readerRenewed(DiscoveryItem discoveryItem) {
    System.out.println("Renew:\n" + discoveryItem.toString());
}

public void readerRemoved(DiscoveryItem discoveryItem) {
    System.out.println("Removed:\n" + discoveryItem.toString());
}
```

Gráfico 27. Network Discovery

2.2.3.5.1. NetworkDiscoveryListenerService Public Methods

Los métodos *public* para *NetworkDiscoveryListenerService* son los mismos que para *SerialDiscoveryListenerService*.

2.2.3.6. Discovery Service Exceptions

Las clases en el paquete *com.alien.enterpriseRFID.discovery* manejan los errores lanzando excepciones hacia el objeto. Hay tres excepciones *discovery*, todo se extiende desde una sola clase *AlienDiscoveryException:*

1. *AlienDiscoverySerialException*
2. *AlienDiscoverySocketException*
3. *AlienDiscoveryUnknownReaderException*

2.2.3.6.1. AlienDiscoverySerialException

Esta excepción puede ser lanzada por un *SerialDiscoveryListenerService* si las clases serial no están presentes.

2.2.3.6.2. AlienDiscoverySocketException

Esta excepción puede ser lanzada por un *NetworkDiscoveryListenerService* si no es capaz de enlazar con el puerto receptor especificado. Esto es probable debido a que otra aplicación tiene vinculados al mismo puerto (tal vez otro de descubrimiento de servicios).

2.2.3.6.3. AlienDiscoveryUnknownReaderException

Esta excepción se produce cuando se intenta crear un objeto *Reader* desde *DiscoveryItem* a través de su método *getReader()*. Si el *ReaderType* contenido en el *DiscoveryItem* es ahora un tipo reconocido, entonces el tipo correcto de objeto *Reader* no puede ser creado, y falla el método.

2.2.4. Notify Classes

2.2.4.1. Introducción

Las clases *Notify* trabajan conjuntamente con un *Reader* que se ejecutan en modo autónomo. El *Reader* está configurado para leer *tags* una y otra vez sin necesidad de interacción humana, puede configurarse para enviar mensajes a los servicios de escucha en la red cuando se producen eventos específicos, como por ejemplo la expiración de un temporizador, adición/eliminación de *tags* de la lista de *tags*, programación exitosa/no exitosa, etc..

Las clases *notify* implementan servicios de escucha, constantemente esperando y escuchando los mensajes de notificación de los *Reader*, y convertir estos mensajes en objetos Java que están disponibles para su aplicación.

La misma clase *notify* que escuchan mensajes de notificación periódica desde el *Reader* también manejará datos *streaming* desde el *Reader* – los modos *Tagstream* y *IOStream* empuja *tags* y eventos *I/O* a los oyentes que se produzcan en el *Reader*, proporcionando datos de baja latencia alcanzables a través del mecanismo de notificación tradicional.

2.2.4.2. MessageListenerService Class

La clase clave en el *Notify Package* es *MessageListenerService*. Este es un servicio que escucha en un puerto específico para mensajes entrantes para el *Reader*. Lo que sigue es el código básico de que muestra cómo utilizar el *MessageListenerService*:

```
import com.alien.enterpriseRFID.notification.*;

public class MessageListenerTest implements MessageListener  {

public MessageListenerTest() throws Exception {
    MessageListenerService service = new MessageListenerService(3988);
    service.setMessageListener(this);
    service.startService();
    ...(application continues)...
}


public void messageReceived(Message message) {
    System.out.println("Message Received: " + message.toString());
}
```

Gráfico 28. Ejemplo de código para el uso de *MessageListenerService*

El *MessageListenerService* está configurado para escuchar un número de puerto especificado, y el *Reader* (o varios *Reader*) se pueden configurar para notificar al servicio en ese puerto. Esto se hace como sigue:

1. Instruir al *Reader* para enviar mensajes de notificación a los host que están ejecutando el *MessageListenerService*. Por ejemplo, si el servicio se ejecuta en una máquina llamada "*listener.alien.com*", el comando *Reader* sería:

   Reader.setNotifyAddress("listener.alien.com:3988");

2. Instruir al *Reader* acerca de las condiciones que debe desencadenar los mensajes de notificación. Por ejemplo, para informar a los *Reader* para enviar una lista de *tags* cada 30 segundos, podría expedirse el siguiente comando:

   Reader.setNotifyTime(30);

3. Configure el formato de mensaje de notificación en XML. En orden para *MessageListenerService* para poder descifrar el mensaje de notificación, debe ser en formato "*Text*" o "XML":

   Reader.setNotifyFormat(Reader.XML_FORMAT);

4. Finalmente decir al *Reader* para comenzar la lectura de las *tags* de modo autónomo. Por ejemplo para informar a los *Reader* a leer tan rápido como pueda hasta que dijo lo contrario, se podrían emitir los siguientes comandos:

Reader.autoModeReset();

Reader.setAutoMode(Reader.ON);

En este punto el *Reader* cambia al modo autónomo y, como se indica, envía un mensaje cada 30 segundos para el *MessageListenerService*, que contiene su lista de *tags* internas e información adicional sobre la notificación.

Cuando se recibe un mensaje, es analizado y convertido en un objeto *Message*, el cual pasa al método *messageReceived*() del *MessageListener* registrado.

2.2.4.2.1. MessageListenerService Public Methods

- ***public int getListenerPort()***

Devuelve el número de puerto a la escucha en los mensajes entrantes de notificación

- ***public void setListenerPort(int listenerPort)***

Especifica el número de puerto que escucha en los mensajes entrantes de notificación

- ***public MessageListener getMessageListener()***

Devuelve el MessageListener registrado para recibir eventos de notificación.

- ***public void setMessageListener(MessageListener messageListener)***

Registra un MessageListener para recibir los eventos de notificación

- ***public void startService()***

Inicia el MessageListenerService.

- ***public void stopService()***

Para el *MessageListenerService*.

- ***public boolean isRunning()***

Devuelve true si el *MessageListenerService* se está ejecutando

- ***public void setIsCustomTagList()***

  Banderas si se usa el decodificador de la lista de *tags* personalizadas o el decodificador estándar de formato "*Text*" al decodificar datos de etiqueta.

2.2.4.3. MessageListener Interface

Para que un objeto reciba eventos de notificación de una *MessageListenerService*, se debe implementar la interfaz *MessageListener*. Esta interfaz se define como sigue:

```
public interface MessageListener{
    public void messageReceived(Message message);
}
```

Gráfico 29. MessageListener

Sólo un método para implementar y simplemente recibe un objeto *Message* de *MessageListenerService.*

2.2.4.4. Message Class

Un objeto *Message* encapsula una colección de metadatos sobre el propio mensaje de notificación y una matriz de objetos *Tag* extraído de la parte de la lista de *tags* del mensaje de notificación. Contiene los siguientes miembros, todos los cuales están disponibles a través de los métodos de acceso *getter y setter*.

- ***ReaderName***– el nombre del *Reader*
- ***ReaderType***– el tipo de *Reader*
- ***IPAddress*** – la dirección IP del *Reader*
- ***MACAddress***– la dirección MAC del *Reader*, si la proporciona
- ***CommandPort***– el número de Puerto en el que enviamos los comandos al *Reader*
- ***Time***– la fecha y hora en que fue enviado el mensaje
- ***Reason***– la razón por la que el *Reader* envió el mensaje
- ***StartTriggerLines***– indica cuál de las entradas externas activa al *Reader* para iniciar
- ***StopTriggerLines***– indica cuál de las entradas externas activa al *Reader* para terminar
- **TagList**– una matriz de objetos *Tag* extraídos de la notificación

2.2.4.4.1. Message Public Methods

- ***public String getReaderName()***

Devuelve el nombre del *Reader* que envió el mensaje de notificación.

- ***public String getReaderType()***

Devuelve el tipo del *Reader* que envió el mensaje de notificación

- ***public String getReaderIPAddress()***

Devuelve la dirección IP del *Reader* que envió el mensaje de notificación.

- ***public int getReaderCommandPort()***

Devuelve el número de puerto de comando del *Reader* que envió el mensaje de notificación.

- ***public String getReaderMACAddress()***

Devuelve la dirección MAC del *Reader* que envió el mensaje de notificación, caso contrario devuelve null.

- ***public Date getDate()***

Devuelve la fecha y hora del mensaje.

- ***public String getReason()***

Obtiene la razón de por qué el *Reader* envía el mensaje.

- ***public int getStartTriggerLines()***

Devuelve las líneas de entrada externas que desencadenó este ciclo autónomo para empezar.

- ***public int getStopTriggerLines()***

Devuelve las líneas de entrada externas que desencadenó este ciclo autónomo para empezar.

- ***public int getTagCount()***

Devuelve el número de *tags* en el *Tag*List del mensaje.

- ***public Tag[] getTagList()***

Devuelve la lista de *tags* o *Tag*List del mensaje de notificación, como una matriz de *Tags*.

- ***public Tag getTag(int index)***

Devuelve la *Tag* que contiene el índice de posición en la lista de *tags* (*Tag*List) del mensaje.

- ***public String getRawData()***

Devuelve el contenido sin formato del mensaje de notificación, antes de la decodificación.

- ***public int getIOCount()***

Devuelve el número de eventos I/O en los mensajes IOList.

- ***public ExternalIO[] getIOList()***

Devuelve el IOList de un mensaje de notificación, como una matriz de ExternalIOs.

- ***public ExternalIO getIO(int index)***

Devuelve el ExternalIO que contiene el índice de posición en la IOList del mensaje

2.2.4.5. ErrorMessage Class

Se utiliza un objeto *ErrorMessage* por el *MessageListenerService* para comunicar cualquier problema que tenga mientras al intenta recibir o descifrar un mensaje de notificación de un *Reader*. *ErrorMessage* se extiende de *Message*, por lo que puede entregarlos al método *MessageReceived*() de la misma manera que *MessageListener*.

La propiedad *ErrorMessage* devolverá información útil sobre el error, con los siguientes métodos:

- ***getReaderIPAddress()***– *Reader* que estaba enviando el mensaje
- ***getReason()***– la razón del error
- ***getRawData()***– los datos que fueron recibidos por el *Reader*

El manejo de estas condiciones, el uso de "*instanceof*" en tu *MessageReceived(),* de la siguiente manera:

```
public void MessageReceived(Message m) {
    if (message instanceof ErrorMessage) {
        // message is bad
        System.out.println("Notify error from " + m.getReaderIPAddress());
        System.out.println("Problem is: " + m.getReason());
        System.out.println("Data read is: " + m.getXML());
    } else {
        // message is good
        ...
    }
}
```

Gráfico 30. Uso de instanceof

2.2.4.6. Message Class Exceptions

Las clases del paquete *com.alien.enterpriseRFID.notify*manejan circunstancias de error lanzando excepciones hacia el objeto de llamada. Hay dos notificaciones de excepción, todo extensible desde una clase *AlienMessageExceptionclass:*

2.2.4.6.1. AlienMessageConnectionException

Puede producir esta excepción si el *Reader* encuentra con un error de comunicación durante la recepción de un mensaje de un *Reader*.

2.2.4.6.2. AlienMessageFormatException

Puede producir esta excepción si el mensaje no está en formato XML, o hay algún problema al analizar el XML.

Capítulo III. Desarrollo de Interfaz RFID

3.1 Análisis

3.1.1 Recopilación de requerimientos

Requisitos Específicos:

R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o la lectura continua de *tags*, según sea el caso de aplicación.

R.1.2 Escritura de la información en la Base de Datos.

R.1.3 Creación de archivos XML con la información que se proporciona de la tecnología RFID.

R.1.4 Registrar los errores que se obtengan del uso de las APIs de la tecnología RFID.

R.1.5 Actualizar la configuración del Comportamiento.

R.1.6 Actualizar la configuración de la Persistencia.

R.1.7 Actualizar la configuración del Hardware.

3.1.2 Descripción de las tecnologías a utilizar

3.1.2.1 Lenguaje de Programación Java

Java es un lenguaje de programación y la primera plataforma informática desarrollada por James Gosling y creada por Sun Microsystems en 1995. Es la tecnología subyacente que permite el uso de programas punteros, como herramientas, juegos y aplicaciones de negocios.

El lenguaje en sí mismo toma mucha de su sintaxis de C, Cobol y Visual Basic, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel.

La versatilidad, eficacia, portabilidad de plataformas y seguridad de la tecnología Java la convierte en la tecnología ideal para la informática de redes. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet.

(Java, 2012)

3.1.2.2 *Enterprise JavaBeans*

Con la tecnología J2EE (*Java 2 Platform, Enterprise Edition*) *Enterprise JavaBeans* es posible desarrollar componentes (*enterprise beans*) que luego puedes reutilizar y ensamblar en distintas aplicaciones.

Con la programación orientada a objetos se puede reutilizar clases, pero con componentes es posible reutilizar un mayor nivel de funcionalidades e incluso es posible modificar estas funcionalidades y adaptarlas a cada entorno de trabajo particular sin tocar el código del componente desarrollado. Se puede ver a un componente como un objeto tradicional con un conjunto de servicios adicionales soportados en tiempo de ejecución por el contenedor de componentes. El contenedor de componentes se denomina *contenedor EJB* y es algo así como el sistema operativo en el que éstos residen. Recuerda que en Java existe un modelo de programación de objetos remotos denominado RMI. Con RMI es posible enviar peticiones a objetos que están ejecutándose en otra máquina virtual Java. Podemos ver un componente EJB como un objeto remoto RMI que reside en un contenedor EJB que le proporciona un conjunto de servicios adicionales.

Los servicios más importantes son los siguientes:

- Manejo de transacciones: apertura y cierre de transacciones asociadas a las llamadas a los métodos del Bean.
- Seguridad: comprobación de permisos de acceso a los métodos del Bean.
- Concurrencia: llamada simultánea a un mismo Bean desde múltiples clientes.
- Servicios de red: comunicación entre el cliente y el Bean en máquinas distintas.
- Gestión de recursos: gestión automática de múltiples recursos, como colas de mensajes, bases de datos o fuentes de datos en aplicaciones heredadas que no han sido traducidas a nuevos lenguajes/entornos y siguen usándose en la empresa.
- Persistencia: sincronización entre los datos del Bean y tablas de una base de datos.
- Gestión de mensajes: manejo de Java Message Service (JMS).
- Escalabilidad: posibilidad de constituir clusters de servidores de aplicaciones con múltiples hosts para poder dar respuesta a aumentos repentinos de carga de la aplicación con sólo añadir hosts adicionales.

- Adaptación en tiempo de despliegue: posibilidad de modificación de todas estas características en el momento del despliegue del bean.

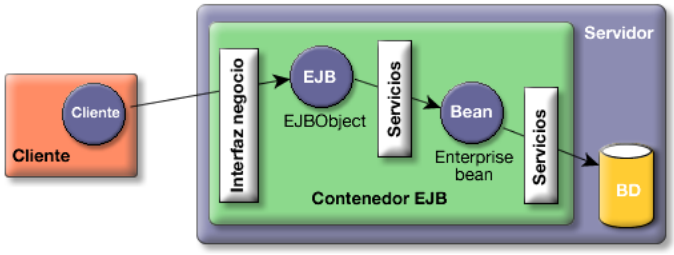


Gráfico 31. Contenedor EJB

Representación de alto nivel del funcionamiento de los Enterprise beans. (JTech, 2003)

3.1.2.3 *Java Persistence API*

Java Persistence API, más conocida por sus siglas JPA, es la API de persistencia desarrollada para la plataforma Java EE.

Es un *framework* del lenguaje de programación Java que maneja datos relacionales en aplicaciones usando la Plataforma Java en sus ediciones *Standard* (Java SE) y *Enterprise* (Java EE).

Persistencia, almacenar los datos de una aplicacion en un medio fisico como una base de datos, un archivo de texto, etc., en este contexto cubre tres áreas:

- La API en sí misma, definida en *javax.persistence.package*
- La *Java Persistence Query Language (JPQL)*
- Metadatos objeto/relacional

El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos (siguiendo el patrón de mapeo objeto-relacional), como sí pasaba con EJB2, y permitir usar objetos regulares (conocidos como POJOs(*Plain Old Java Object*) son clases que no extienden a ninguna otra).

(SlideShare, 2010)

3.1.2.4 XML

XML son las siglas de*Extensible Markup Language*, una especificación/lenguaje de programación desarrollada por el *World Wide Web Consortium* o W3C. XML es una versión de SGML (*Standard Generalized Markup Language* o "Estándar de Lenguaje de Marcado Generalizado"), diseñado especialmente para los documentos de la web. Permite que los diseñadores creen sus propias etiquetas, permitiendo la definición, transmisión, validación e interpretación de datos entre aplicaciones y entre organizaciones.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

(W3C, 2008)

3.1.2.5 *JavaServer Faces*

JavaServer Faces (JSF) es una tecnología y *framework* para aplicaciones Java basadas en web que simplifica el desarrollo de *Interface*s de usuario en aplicaciones Java EE. JSF usa JavaServer Pages (JSP) como la tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías como XUL (acrónimo de XML-based User-*Interface* Language, lenguaje basado en XML para la interfaz de usuario).

JSF incluye:

- Un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad.
- Un conjunto por defecto de componentes para la interfaz de usuario.
- Dos bibliotecas de etiquetas personalizadas para JavaServer Pages que permiten expresar una interfaz JavaServer Faces dentro de una página JSP.
- Un modelo de eventos en el lado del servidor.
- Administración de estados.

- Beans administrados.

Características principales:

La tecnología *JavaServer Faces* constituye un marco de trabajo (*framework*) de *Interface*s de usuario del lado de servidor para aplicaciones web basadas en tecnología Java y en el patrón MVC (Modelo Vista Controlador).

Los principales componentes de la tecnología JavaServer Faces son:

- ❖ Una API y una implementación de referencia para:
  - Representar componentes de interfaz de usuario (UI-*User Interface*) y manejar su estado.
  - Manejar eventos, validar en el lado del servidor y convertir datos
  - Definir la navegación entre páginas
  - Soportar internacionalización y accesibilidad, y proporcionar extensibilidad para todas estas características.
  - Una librería de etiquetas *JavaServer Pages* (JSP) personalizadas para dibujar

Este modelo de programación bien definido y la librería de etiquetas para componentes UI facilita de forma significativa la tarea de la construcción y mantenimiento de aplicaciones web con UIs en el lado servidor. Con un mínimo esfuerzo, es posible:

• Conectar eventos generados en el cliente a código de la aplicación en el lado servidor.

• Mapear componentes UI a una página de datos en el lado servidor.

• Construir una interfaz de usuario con componentes reutilizables y extensibles.

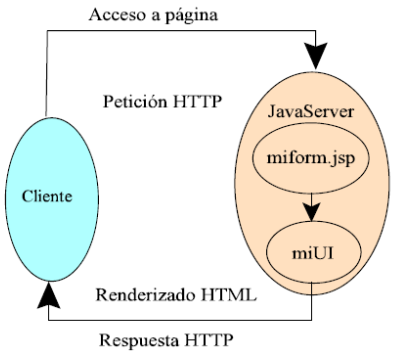
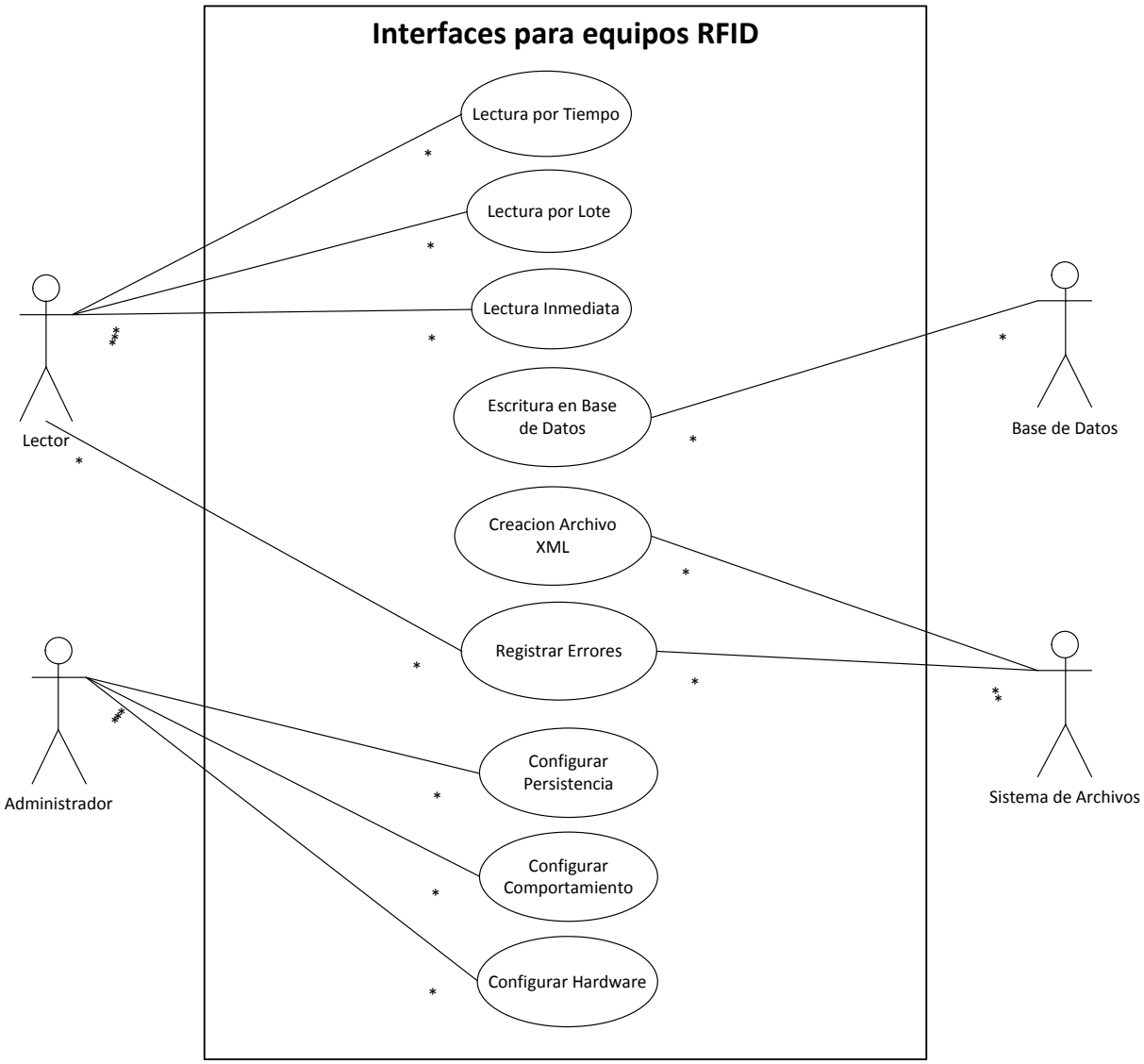


Gráfico 32. Diagrama de una aplicación JSF

(Sicuma, 2011)

3.1.3 Documentación de requerimientos

3.1.3.1 Diagrama de Casos de Uso



3.1.3.2 Descripción de los Casos de Uso

Caso de Uso 1:	Lectura por Tiempo
Actor:	Lector
Descripción:	Se realiza la lectura indicando las Horas, Minutos, Segundos y Milisegundos que realiza su lectura.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o lectura continua de *tags*, según sea el caso de aplicación.	

Caso de Uso 2:	Lectura por Lote
Actor:	Lector
Descripción:	Se realiza según la cantidad de *tags* que se indique.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o lectura continua de *tags*, según sea el caso de aplicación.	

Caso de Uso 3:	Lectura Inmediata
Actor:	Lector
Descripción:	Se realiza de forma continua.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o lectura continua de *tags*, según sea el caso de aplicación.	

Caso de Uso 4:	Escritura en Base de Datos
Actor:	Base de Datos
Descripción:	Se realiza la escritura de la información leída directamente en la Base de Datos.
Prioridad:	Obligatorio

Requisitos Asociados
R.1.2 Escritura de la información en cualquier Base de Datos.
R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o la lectura continua de *tags*, según sea el caso de aplicación.

Caso de Uso 5:	Creación de Archivo XML
Actor:	Sistema de Archivos
Descripción:	Se crea un archivo XML con la información leida.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.3 Creación de archivos XML con la información que se proporciona de la tecnología RFID.	
R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o la lectura continua de *tags*, según sea el caso de aplicación.	

Caso de Uso 6:	Registrar Errores
Actor:	Sistema de Archivos
Descripción:	Se registrarán los errores que obtengamos del sistema RFID.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.4 Registrar los errores que se obtengan del uso de las APIs de la tecnología RFID.	
R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o la lectura continua de *tags*, según sea el caso de aplicación.	

Caso de Uso 7:	Configurar Persistencia
Actor:	Administrador
Descripción:	Se realiza la configuración de la Persistencia en la Base de Datos como en el Archivo XML.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.6 Actualizar la configuración de la Persistencia.	
R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o la lectura continua de *tags*, según sea el caso de aplicación.	

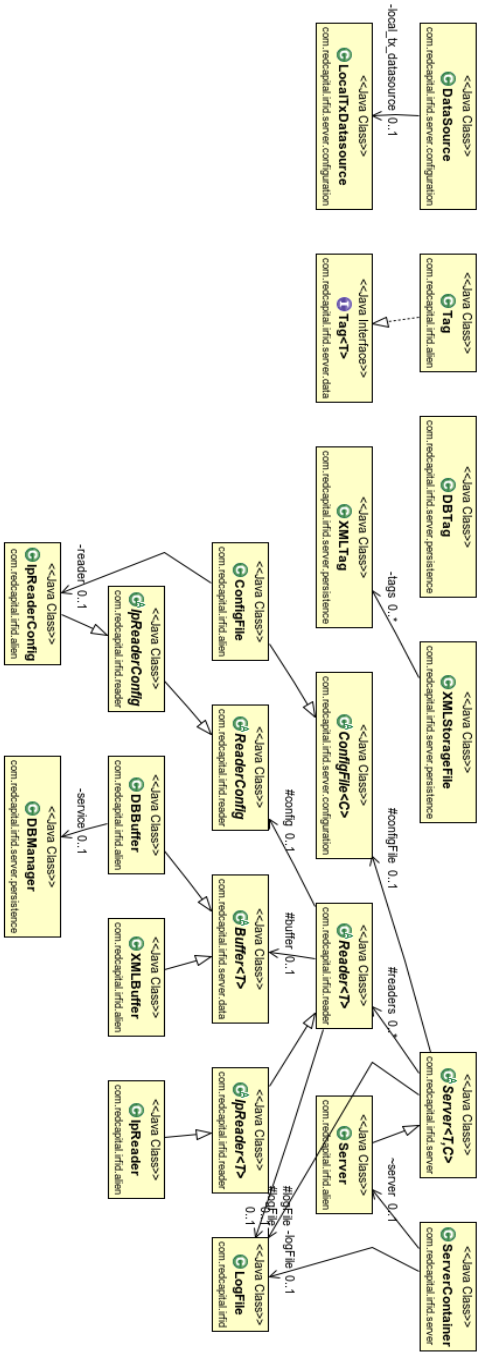
Caso de Uso 8:	Configurar Comportamiento
Actor:	Administrador
Descripción:	Se realiza la configuración del comportamiento de las lecturas pueden ser por Tiempo, Lotes o Inmediata.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.5 Actualizar la configuración del Comportamiento.	
R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o la lectura continua de *tags*, según sea el caso de aplicación.	

Caso de Uso 9:	Configurar Hardware
Actor:	Administrador
Descripción:	Se realiza la configuración del lector la dirección IP del lector, nombre, usuario y contraseña, con el que se va a trabajar.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.7 Actualizar la configuración del Hardware.	

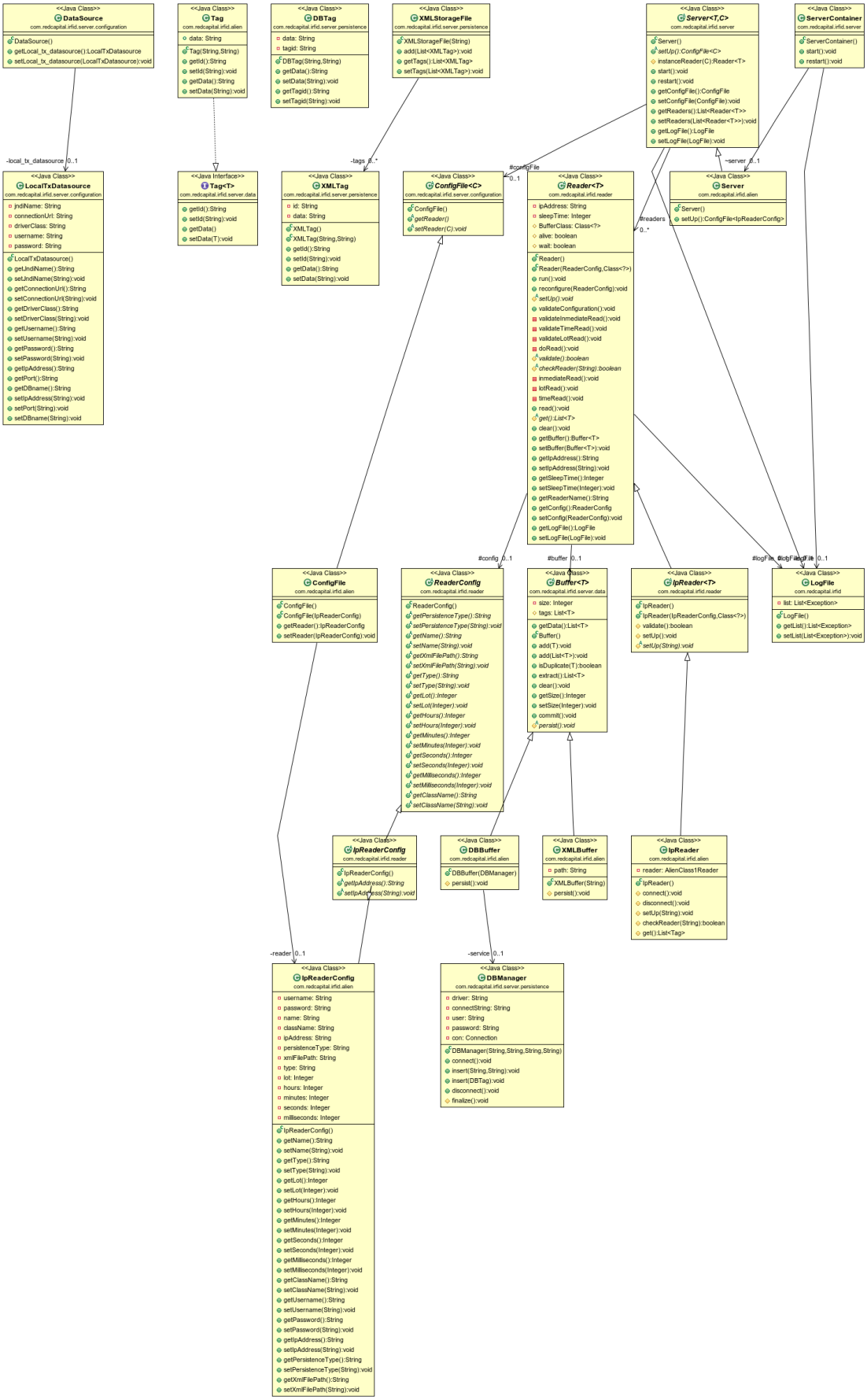
Caso de Uso 10:	Configuración Completa
Actor:	Administrador
Descripción:	Se realizan todas las configuraciónes Persistencia, Comportamiento, Hardware en un solo paso para obtener una API completa.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.1 Permitir la lectura de *tags*, esto sea cada cierto Tiempo, por número de *tags* Lotes, o la lectura continua de *tags*, según sea el caso de aplicación.	
R.1.2 Escritura de la información en cualquier Base de Datos.	
R.1.3 Creación de archivos XML con la información que se proporciona de la tecnología RFID.	
R.1.5 Actualizar la configuración del Comportamiento.	
R.1.6 Actualizar la configuración de la Persistencia.	
R.1.7 Actualizar la configuración del Hardware.	

3.2 Diseño

3.2.1 Modelo Conceptual

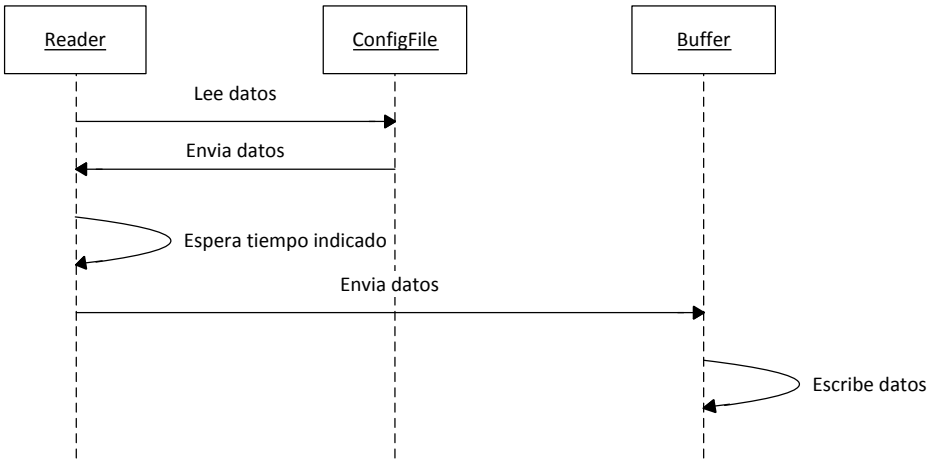


2.2.4. Diagrama de Clases

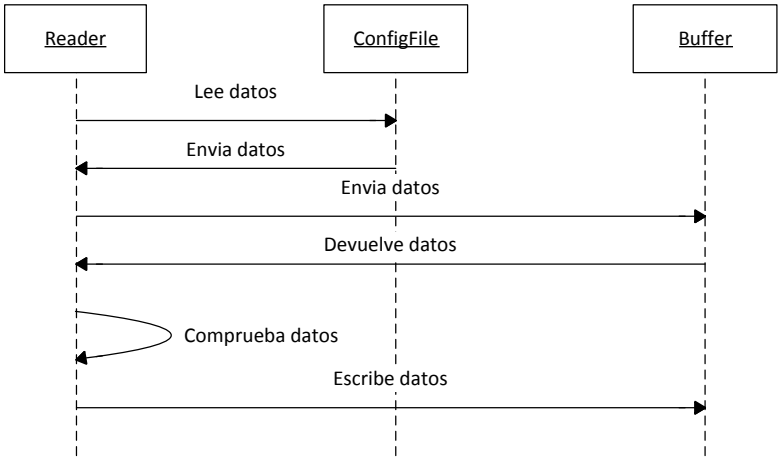


2.2.5. Diagrama de Secuencias

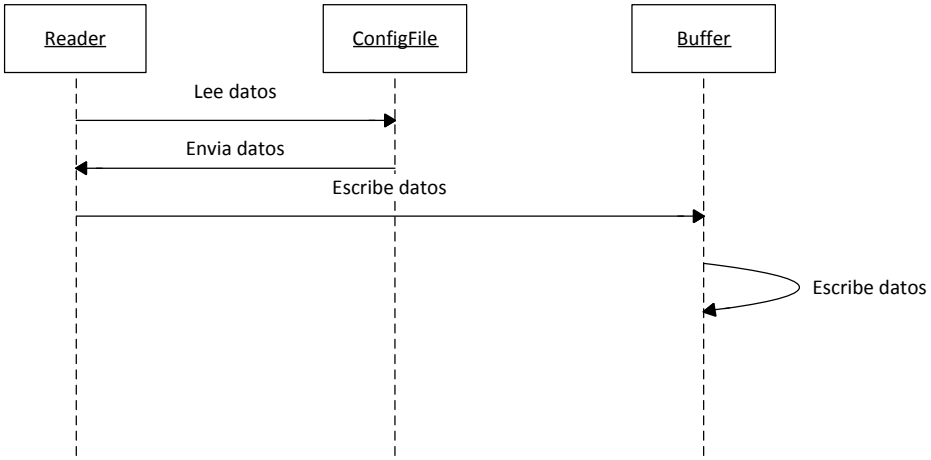
Lectura por Tiempos



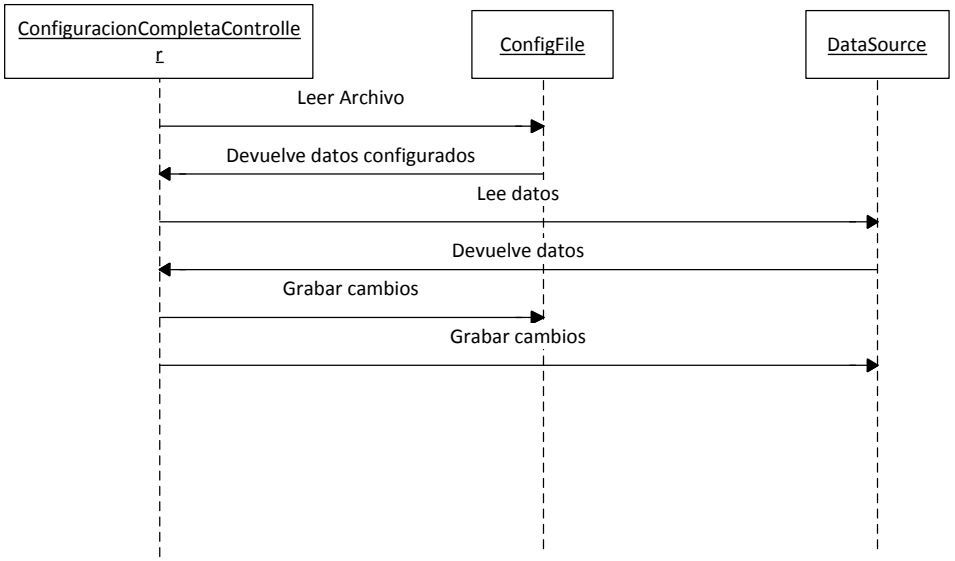
Lectura por Lotes



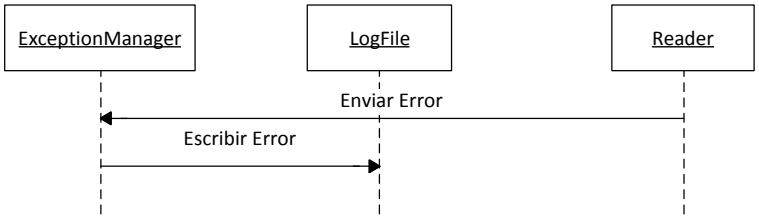
Lectura Inmediata



Configuracion Completa

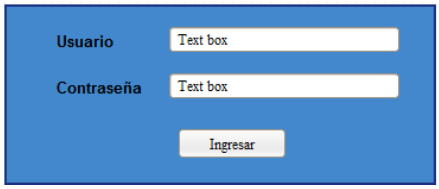


Registro de Errores

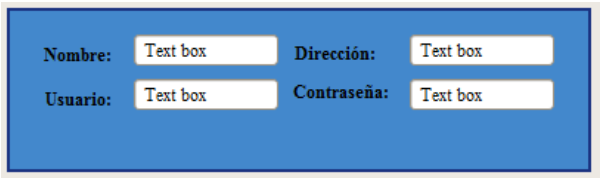


2.2.6. Diseño de *Interfaces*

2.2.6.6. Registro de Usuarios:

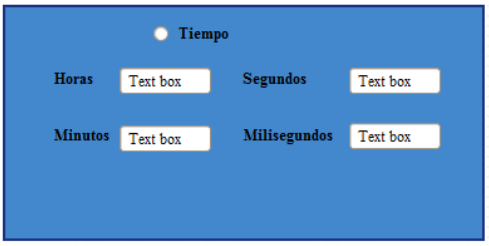


2.2.6.7. Configuración de Lector

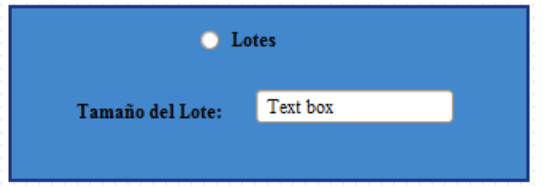


2.2.6.8. Configuración de Comportamiento

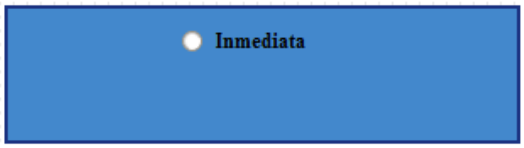
2.2.6.8.1. Configuración Lectura por Tiempo



2.2.6.8.2. Configuración Lectura por Lotes

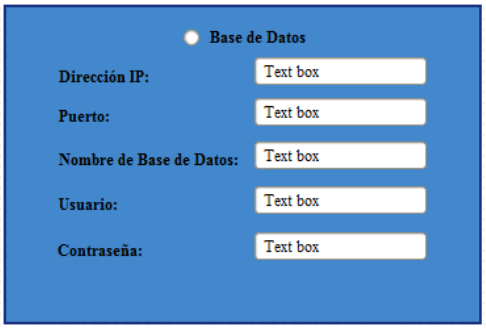


2.2.6.8.3. Configuración Lectura Inmediata

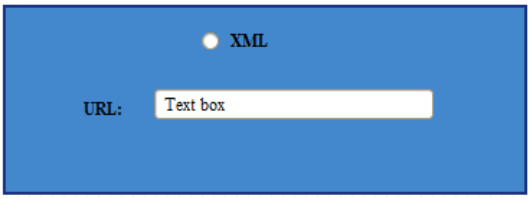


2.2.6.9. Configuración Persistencia

2.2.6.9.1. Configuración de Base de Datos



2.2.6.9.2. Configuración de Archivo XML



3.3 Codificación

3.3.1 Construcción de archivos XML para intercambio de datos de los equipos RFID marca ALIEN.

La creación y escritura del archivo XML se hace con ayuda del framework simplexml mediante el siguiente código:

```
@Root
publicclass XMLStorageFile extends XMLFile {

	@ElementList
	private List<XMLTag>tags;

	publicvoid add(List<XMLTag>tags) {
		if (tags != null) {
			if (!tags.isEmpty()) {
				try {
					load();
					this.tags.addAll(tags);
					write();
				} catch (java.io.FileNotFoundException pe) {
					try {
						this.tags = new ArrayList<XMLTag>();
						this.tags.addAll(tags);
						write();
					} catch (Exception e) {
						// TODO Auto-generated catch block
						e.printStackTrace();
					}
				} catch (Exception e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
			}
		}
	}

}


@Element(name="tag")
publicclass XMLTag {

	@Element(required=false,type=String.class)
	private String id;

	@Element(required=false,type=String.class)
	private String data;

	public XMLTag() {
		super();
		// TODO Auto-generated constructor stub
	}
```

```
	public XML_Tag_(String id, String data) {
		super();
		this.id = id;
		this.data = data;
	}
}
```

En caso de un cambio en la estructura de los datos de las etiquetas que se vayan a implementar se debería sobre escribir la clase XML*Tag* para poder adicionar más campos según sea la estructura de datos de las etiquetas que se vayan a utilizar.

1.3.2 Software para el envío de información hacia una base de datos.

Para la persistencia en base de datos utilizamos el driver jdbc de la base de datos (en este caso Postgres) para la inserción de los datos.

```
	publicvoid insert(ObjectObject) throws Exception {

		PreparedStatement pst = null;

		try {
			pst = prepareStatement(Object);
			pst.executeUpdate();

		} catch (SQLException ex) {
			throw ex;
		} finally {
			if (pst != null)
				pst.close();

		}
	}
```

En caso de un cambio en la estructura de los datos de las etiquetas que se vayan a implementar se debería sobre escribir el método prepareStatement(*ObjectObject*) en una clase que herede de la clase DBManager para poder adicionar mas campos según sea la estructura de datos de las etiquetas que se vayan a utilizar.

```
	protected PreparedStatement prepareStatement(Object data)
			throws SQLException, Exception {
		PreparedStatement pst = null;
		String stm = "INSERT INTO irfid(tagid, data) VALUES(?, ?)";
		pst = con.prepareStatement(stm);
		pst.setString(1, ((Tag)data).getTagID());
		pst.setString(2, ((Tag)data).getData());
		return pst;
	}
```

1.3.3 Consola de control de mensajes y errores.

Se almacena en una clase ApplicationScoped, las excepciones privateLogFile logFile; la cual puede ser accedida por toda la aplicación dando la facilidad de almacenar las excepciones generadas durante la ejecución, con una instrucción simple:

```
try {
} catch (Exception e) {
        logFile.getList().add(e);
}
```

1.3.4 Plataforma de configuración de procesamiento de información por lotes, tiempo e inmediato

Mediante el *framework* simple xml formamos la clase de archivo de configuración, lo leemos y lo escribimos mediante esta herramienta:

```
@Root
publicclass ConfigFile extends
        com.redcapital.irfid.server.configuration.ConfigFile<Ip*Reader*Config> {


    @Element(required=false,type=Ip*Reader*Config.class)
    private Ip*Reader*Config *Reader*;


    public ConfigFile() {
        super();
    }



    public ConfigFile(Ip*Reader*Config *Reader*) {
        super();
        this.*Reader* = *Reader*;
    }

    public Ip*Reader*Config get*Reader*() {
        return*Reader*;
    }


    publicvoid set*Reader*(Ip*Reader*Config *Reader*) {
        this.*Reader* = *Reader*;
    }


    @Override
    protected String obtainPath() {
```

```
            return getClass().getClassLoader().getResource("irfid.xml").getPath();
        }


}
```

1.3.5 Desarrollo de Interfaz con Hardware

Mediante la herencia con las clases del paquete com.redcapital.irfid.*Reader* (desarrollados en esta tesis), obtenemos ya el comportamiento y las propiedades necesarias para la comunicación con el software, codificando únicamente el algoritmo de la lectura particular para la lectura de equipos ALIEN.

```
@Override
        public List<Tag> get() throws Exception {
                        List<Tag>tags = new ArrayList<Tag>();
                        com.alien.enterpriseRFID.tags.Tag[] alienTagList;
                        connect();
                        alienTagList = Reader.getTagList();
                        disconnect();
                        if (alienTagList != null) {
                                for (int i = 0; i < alienTagList.length; i++) {
                                com.alien.enterpriseRFID.tags.Tagtag = alienTagList[i];
                                        tags.add(newTag(tag.getTagID(), "Data " + i));
                                }
                                returntags;
                        } else
                                returnnew ArrayList<Tag>();

        }
```

1.3.6 Desarrollo de Interfaz de conexión entre módulos

La clase encargada de la conexión entre módulos es ServerContainer, la cual es accedida desde el contenedor Enterprise Java Beans desde las clases de la Interfaz gráfica.

```
        publicvoid start() {
                logFile =
(LogFile)FacesContext.getCurrentInstance().getExternalContext().getApplicationMap().get(
"logFile");
                try {
                        if (server != null){
                                server.restart();
                                running = true;
                                showInfo("Servidor reiniciado correctamente");
```

```
                    System.out.println("-------------------------------RESTART-------------------------------");
                    }else {
                        System.out.println("-------------------------------START-------------------------------");
                        //CAMBIO DE ESTANDARIZACION DE SERVIDOR
                        server = new com.redcapital.labcontrol.Server();
                        //-------------------------------------------------
                        server.setLogFile(logFile);
                        server.start();
                        showInfo("Servidor iniciado correctamente");
                        running = true;
                    }
                } catch (Exception e) {
                    logFile.getList().add(new Exception("Start Exception ",new Throwable(e.toString())));
                    showError("Error al iniciar el servidor");
                    running = false;
                }
            }
```

3.3.8 Desarrollo Interfaz Gráfica

La interfaz gráfica está construida con el Modelo Vista-Controlador utilizando el acceso EJB y el *framework JSF*.

El desarrollo tiene como base la clase *SessionController* que tiene como tarea coordinar las tareas de Inicio de sesión y validación de usuarios, así como la clase UsuarioController que es la encargada de manejar el mantenimiento de usuario para la administración del servidor.

1.4 Pruebas

2.2.4. Pruebas de integración

Las pruebas se realizaron en primera instancia con un algoritmo simple que genera valores para simular una lectura en las clases genéricas.

```
            //ALGORITMO DE PRUEBAS SIN LECTOR-------------------------
            List<Tag> testList = new ArrayList<Tag>();
            testList.add(new Tag("1", "DATA"));
            testList.add(new Tag("2", "DATA"));
            testList.add(new Tag("3", "DATA"));

            return testList;
            //ALGORITMO DE PRUEBAS SIN LECTOR-------------------------/
```

Posteriormente se implementa el código de lectura de ALIEN en la clase Ip*Reader* del paquete com.redcapital.irfid.alien para las pruebas ya con el equipo.

```
//ALGORITMO VALIDO------------------------------
                        List<Tag>tags = new ArrayList<Tag>();
                        com.alien.enterpriseRFID.tags.Tag[] alienTagList;
                        System.out.println("TYPE:        "+((IpReaderConfig)
config).getType());
                        connect();
                        alienTagList = Reader.getTagList();
                        disconnect();
                        if (alienTagList != null) {
                        System.out.println("READED " + alienTagList.length + "TAGS");
                                for (int i = 0; i < alienTagList.length; i++) {
                                com.alien.enterpriseRFID.tags.Tagtag = alienTagList[i];
                                        tags.add(newTag(tag.getTagID(), "Data " + i));
                                }
                                returntags;
                        } else
                                returnnew ArrayList<Tag>();
```

Durante el proceso de pruebas se realizaron mantenimiento de errores principalmente en las pruebas con el código de ALIEN ya que se encontró que el lector no acepta una conexión inicial y varias lecturas, ya que tiene que abrir y cerrar la conexión por cada lectura. De lo contrario el lector no devuelve ningún error, pero tampoco devuelve un valor.

CAPITULO IV. Aplicación de la Tecnología

4.1 Análisis

4.1.1 Recopilación de requisitos

Requerimientos Específicos:

R.1.1 Registrar ingreso de personas a los laboratorios.

R.1.2 Registrar salida de personas de los laboratorios.

R.1.3 Crear consulta de bitácora por persona.

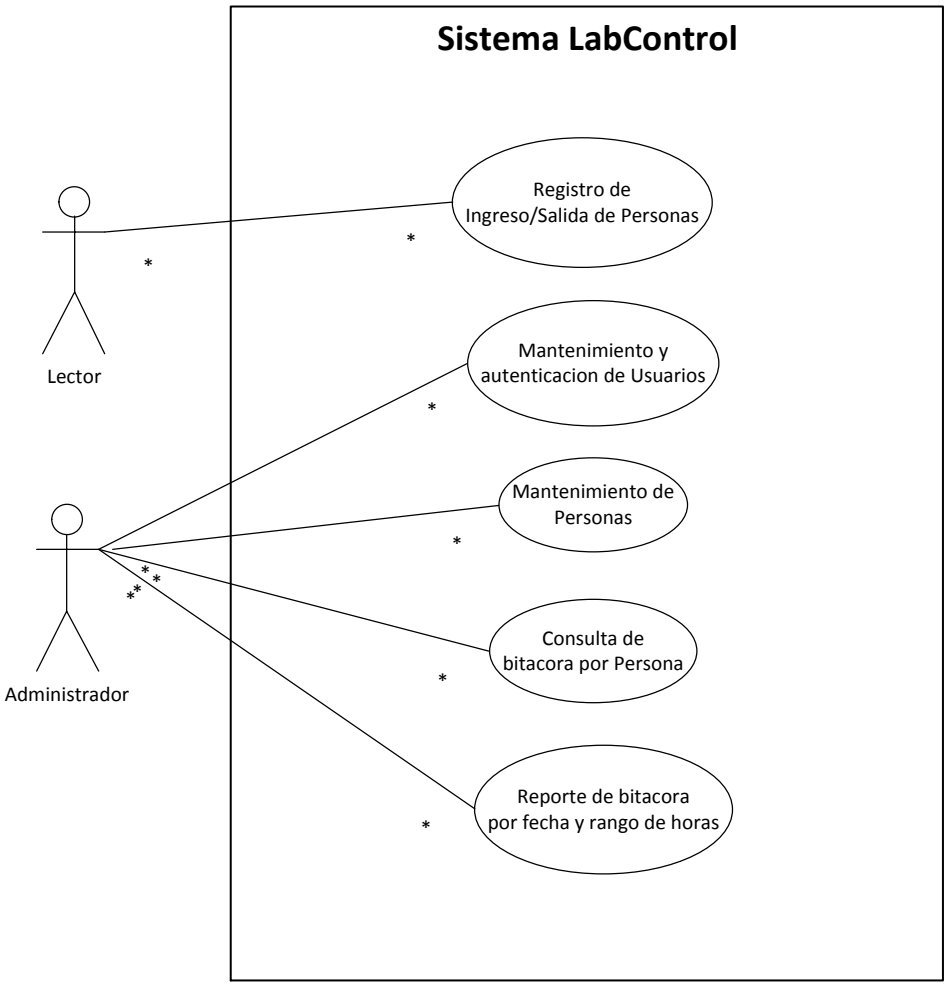
R.1.4 Mantenimiento de personas.

R.1.5 Reporte de bitácora por fecha y rango de horas.

R.1.6 Mantenimiento y autenticación de usuarios.

4.1.2 Documentación de requerimientos

4.1.2.1 Diagrama de Casos de Uso



4.1.2.2 Descripción de los Casos de Uso

Caso de Uso 1:	Registro de Ingreso/Salida de Personas
Actor:	Lector
Descripción:	Se realiza el registro de ingreso o salida de Personas guardando, el nombre, día, y hora.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.1 Registrar ingreso de personas a los laboratorios.	
R.1.2 Registrar salida de personas a los laboratorios.	
R.1.4 Mantenimiento de personas.	

Caso de Uso 2:	Consulta y Reporte de bitácora por Persona
Actor:	Administrador
Descripción:	Marca los ingresos y salidas de una persona en un rango de fechas determinado.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.1 Registrar ingreso de personas a los laboratorios.	
R.1.2 Registrar salida de personas a los laboratorios.	
R.1.3 Crear consulta de bitácora por persona.	
R.1.4. Mantenimiento de personas.	

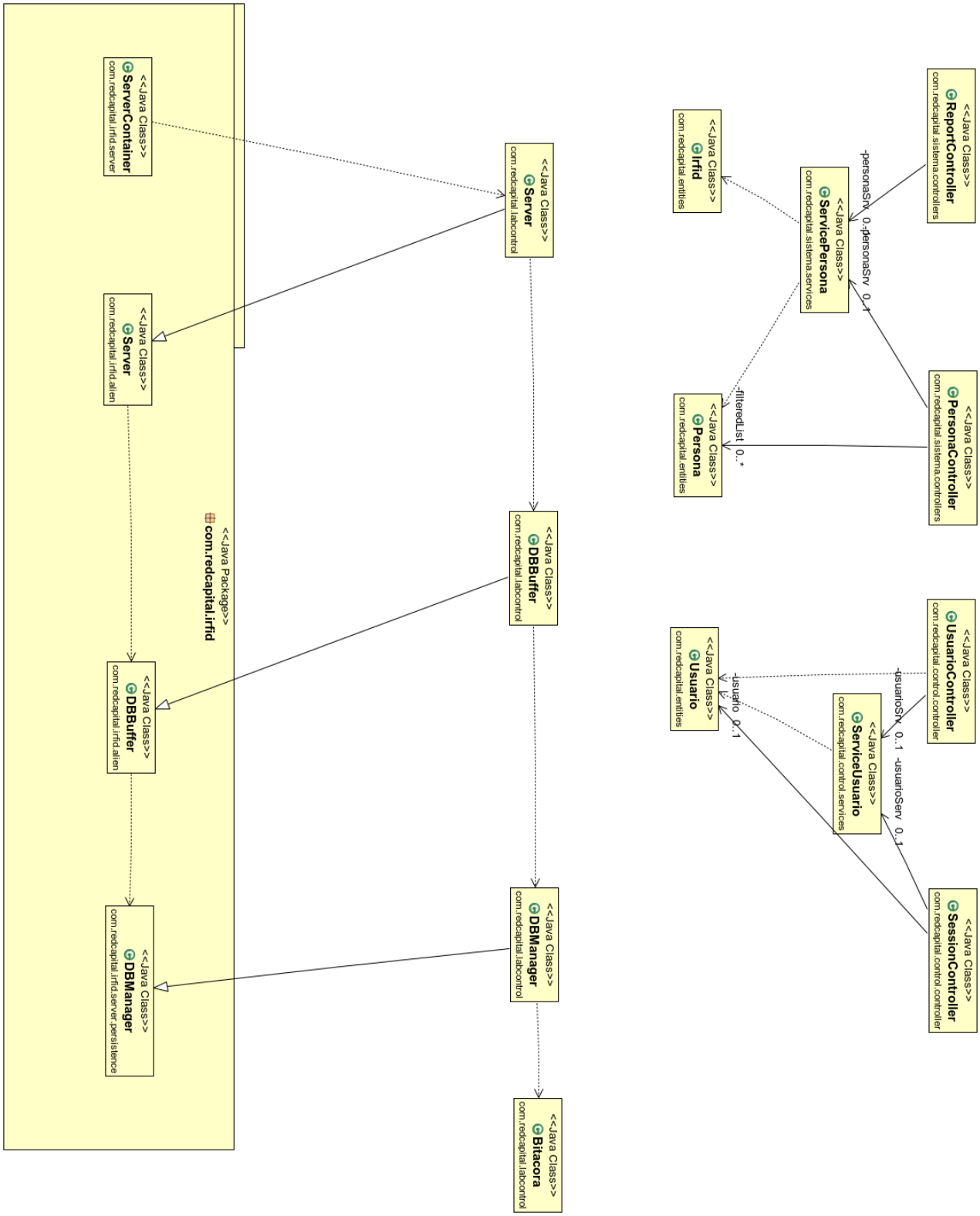
Caso de Uso 3:	Mantenimiento de personas
Actor:	Administrador
Descripción:	Se realiza la lectura de *tags* que se encuentran dentro del alcance y se asignan a cada persona registrada.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.1 Registrar ingreso de personas a los laboratorios.	
R.1.2 Registrar salida de personas de los laboratorios.	
R.1.4 Mantenimiento de personas.	

Caso de Uso 4:	Reporte de control de asistencia general
Actor:	Administrador
Descripción:	Se obtiene un reporte de personas que registraron ingresos y salidas por un determinado rango de fechas y horas.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.1 Registrar ingreso de personas a los laboratorios.	
R.1.2 Registrar salida de personas de los laboratorios.	
R.1.4 Mantenimiento de personas.	
R.1.5 Reporte de bitácora por fecha y rango de horas.	

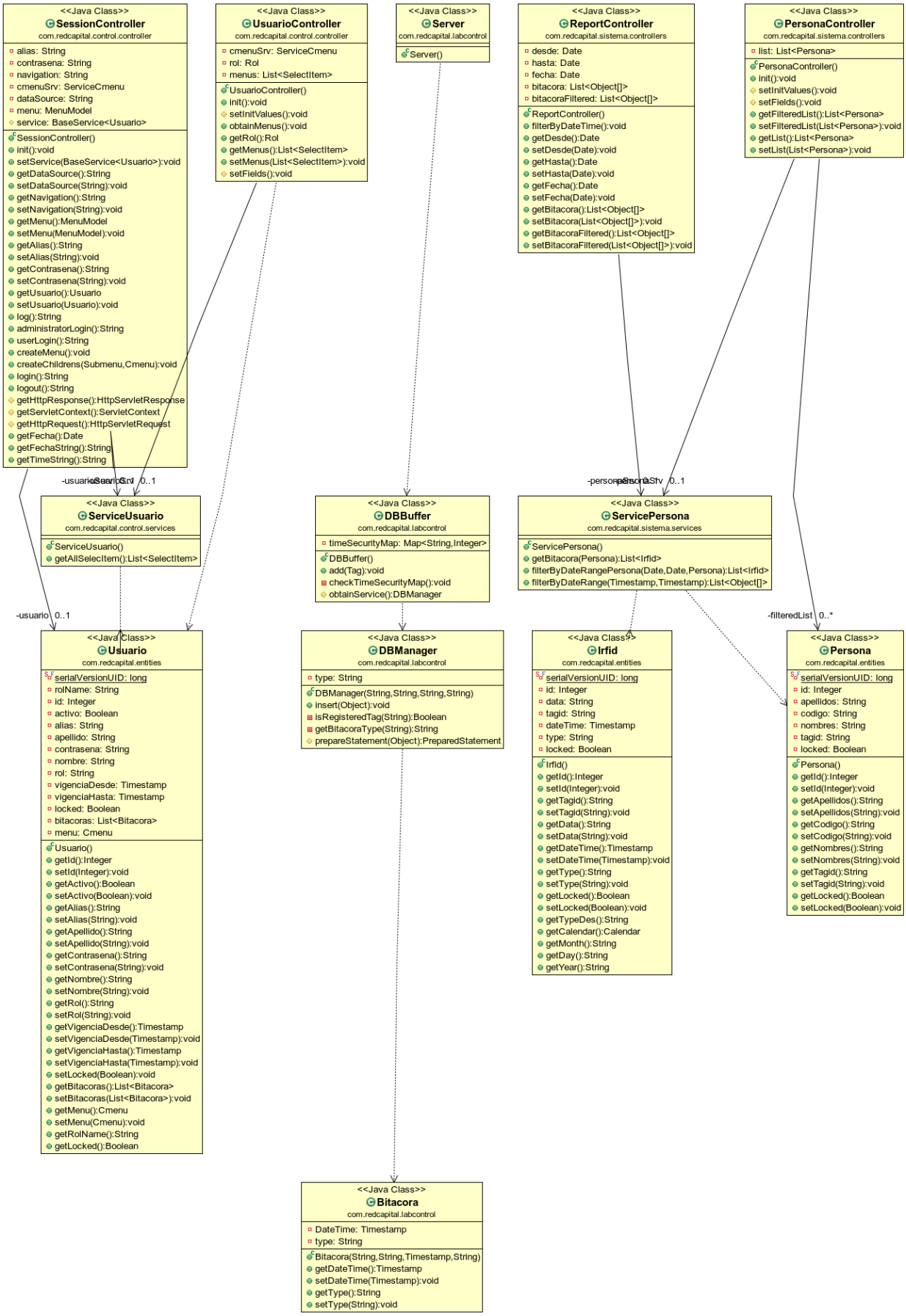
Caso de Uso 5:	Mantenimiento y autenticación de usuarios
Actor:	Administrador
Descripción:	Se realiza el mantenimiento de usuarios.
Prioridad:	Obligatorio
Requisitos Asociados	
R.1.6 Mantenimiento y autenticación de usuarios.	

4.2 Diseño

4.2.1 Modelo Conceptual

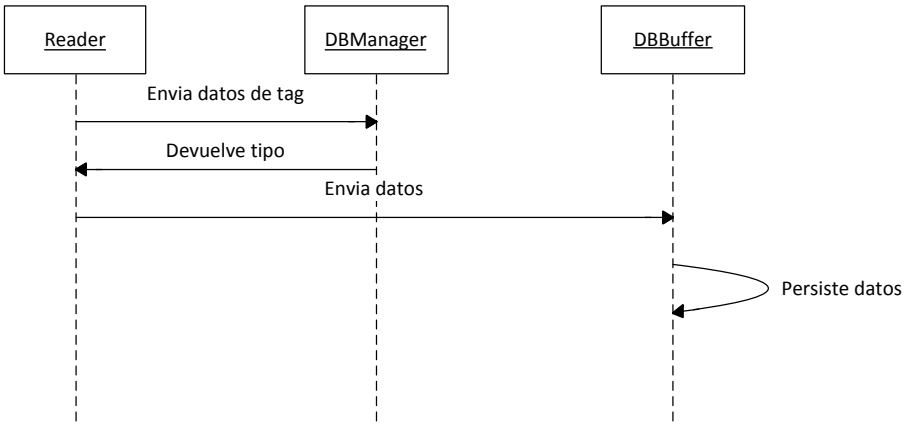


4.2.2 Diagrama de Clases

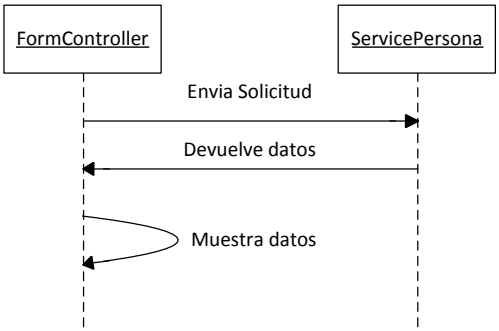


4.2.3 Diagrama de Secuencias

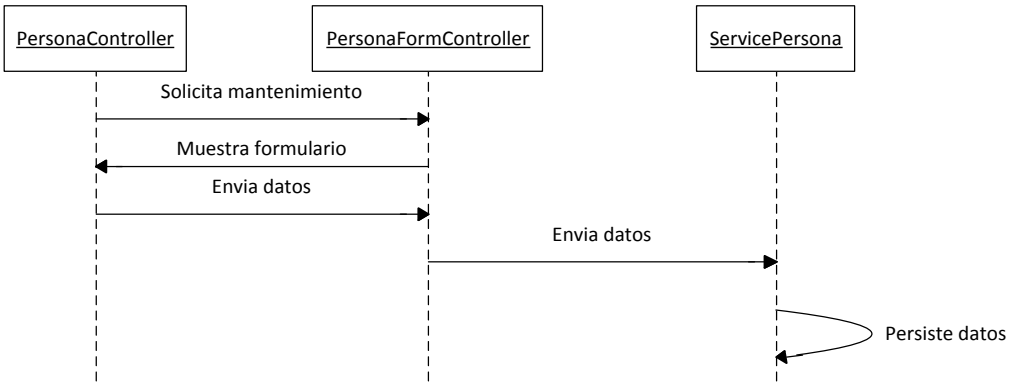
Registro de Ingreso / Salida de Personas



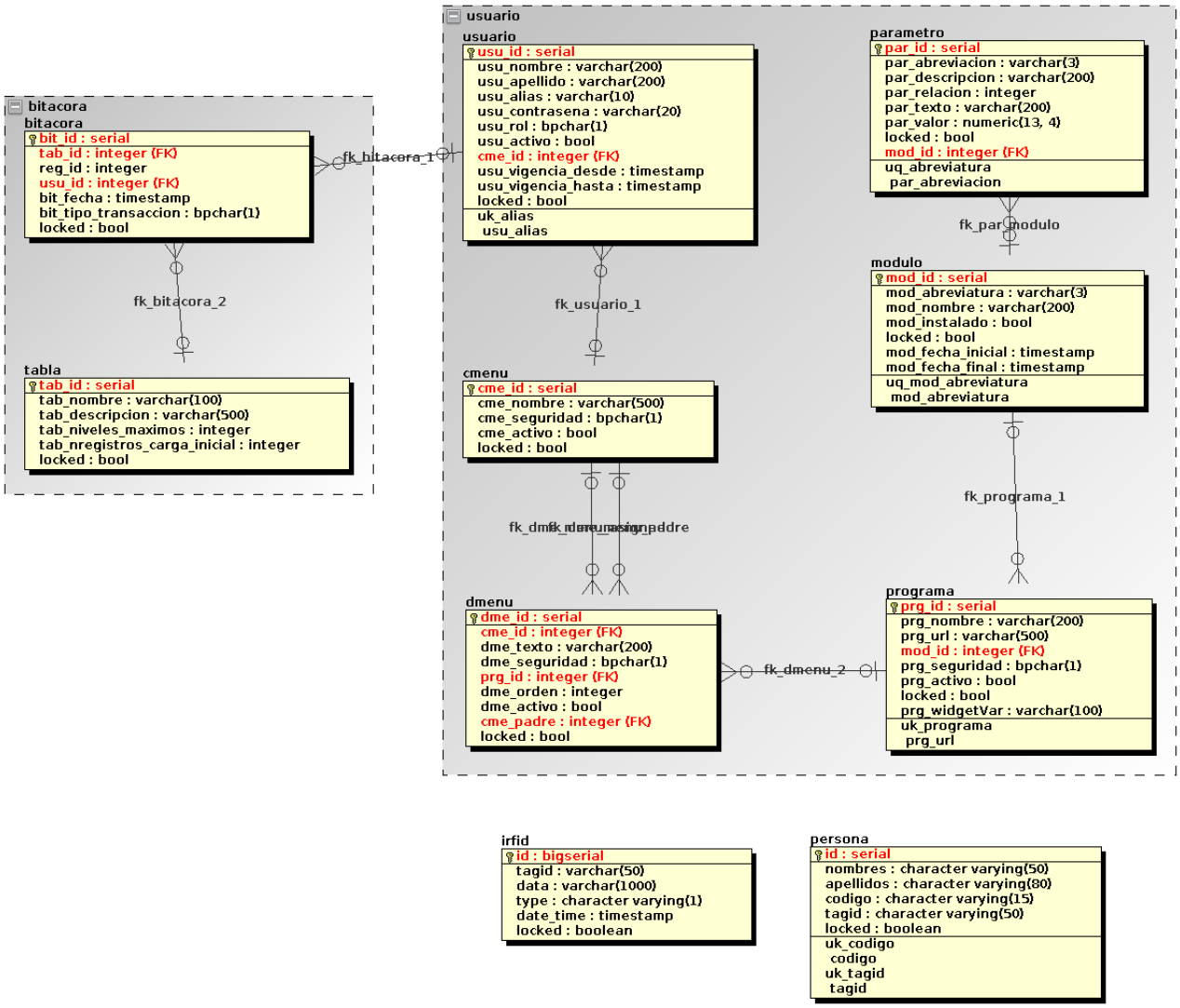
Consulta por Persona



Mantenimiento Persona

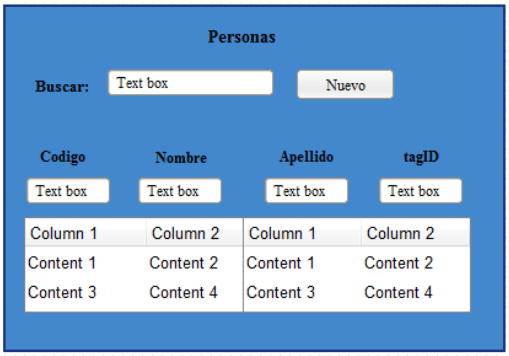


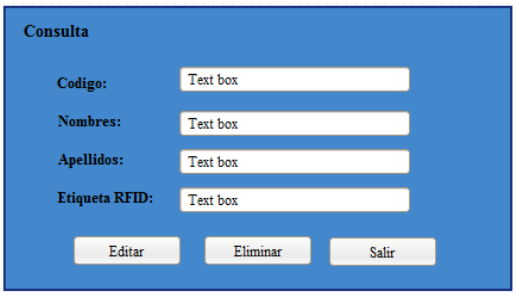
2.2.5. Diagrama Entidad-Relación

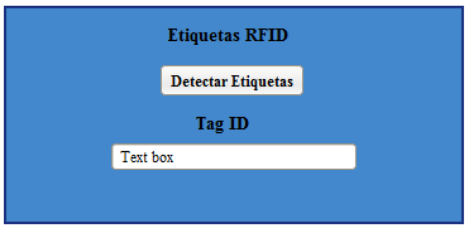


2.2.6. Diseño de *Interfaces*

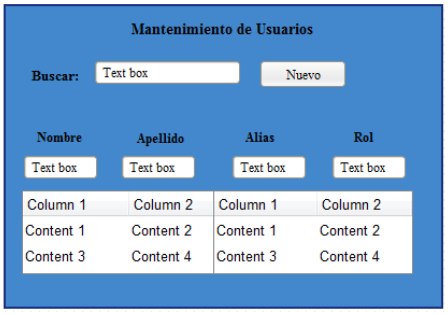
4.2.5.1. Mantenimiento de Personas

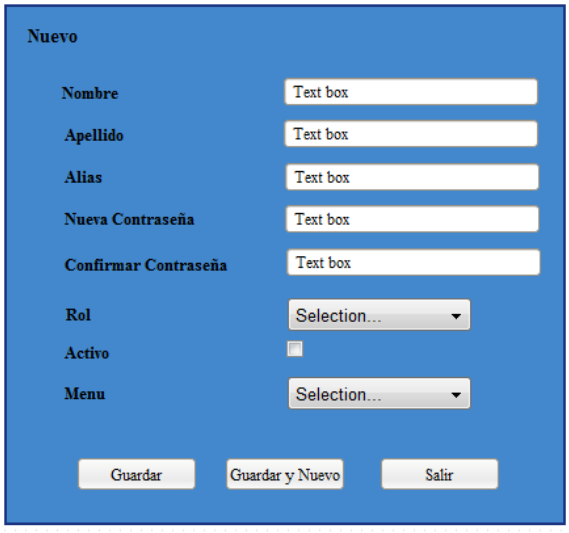




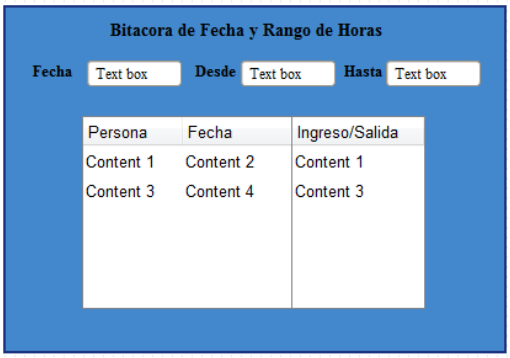


4.2.5.2.Mantenimiento de Usuarios

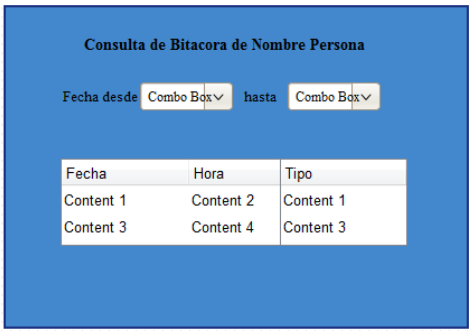




4.2.5.3. Reporte de control de asistencia general



4.2.5.4. Reporte de control de asistencia por persona



4.2.6. Diseño de Arquitectura del Sistema

La arquitectura que vamos a emplear está basada en la arquitectura de tres capas, apoyándonos en el patrón de diseño Modelo Vista Controlador

El Enterprise Java Beans será el encargado del manejo de la parte de Modelo de la aplicación y Postgresql como servidor de persistencia; para la Vista se empleó Java

Server Faces, PrimeFaces, HTML, JavaScript y CSS; además de Jboss en conjunto con Enterprise Java Beans como servidor de Controladores.

4.3 Codificación

4.3.1. Mantenimiento y autenticación de Usuarios

Para la codificación de este caso de uso, se empleó el *frameworkJava Persistence API* para el mapeo con la base de datos y la persistencia.

Las clases codificadas fueron UsuarioController, ServiceUsuario y Usuario. Para la autenticación se codificó la clase *SessionController* que se encarga de controlar los usuarios que tienen acceso a la aplicación para configuración y uso.

4.3.2. Registro de Ingresos y Salidas

Para la codificación de este caso de uso, se empleó el servidor desarrollado en el capítulo III, sobre escribiendo la clase *DBBuffer, Tag y DBManager* para obtener un comportamiento diferente y también agregar los datos adicionales en la persistencia para indicar la fecha, hora y tipo del evento que se registra, además se crea una clase Server que hereda de la clase Server (Capítulo III) que utiliza las nuevas clases con el comportamiento y estructura cambiados.

```
publicclass DBBuffer extends com.redcapital.irfid.alien.DBBuffer {
    private Map<String, Integer> timeSecurityMap;
    public DBBuffer() throws Exception {
            super();
            timeSecurityMap = new HashMap<String, Integer>();
    }
    @Override
    publicvoid add(Tagtag) throws Exception {
            checkTimeSecurityMap();
            if (!timeSecurityMap.containsKey(tag.getId())){
                    super.add(tag);
                    System.out.println("TAG ADDED TO BUFFER");
                    timeSecurityMap.put(tag.getId(),5);
            }else{
                    System.out.println("TAG IGNORED TO BUFFER");
                    thrownew Exception("Time Security Map Exception",new
Throwable("La tag se encuentra en el buffer de seguridad de 5 segundos"));
            }

    }
```

```
privatevoid checkTimeSecurityMap() {

		if (timeSecurityMap != null) {
			for (Map.Entry<String, Integer> entry :
timeSecurityMap.entrySet()) {
				System.out.println("Entry: " + entry.getKey() +"  ,
" + entry.getValue()  );
				if (entry.getValue() > 0)
					entry.setValue(((Integer) entry.getValue()) -
1);
				else
					timeSecurityMap.remove(entry.getKey());

			}
		}else
			timeSecurityMap = new HashMap<String, Integer>();
	}

	@Override
	protected DBManager obtainService() throws Exception {
		DataSource ds = newDataSource();
		ds.load();
		returnnew DBManager(ds.getLocal_tx_datasource()
				.getDriverClass(), ds.getLocal_tx_datasource()
				.getConnectionUrl(),
ds.getLocal_tx_datasource().getUsername(),
				ds.getLocal_tx_datasource().getPassword());
	}
}
```

4.3.3. Mantenimiento de Personas

Para la codificación de este caso de uso, se empleó el *framework Java Persistence API* para el mapeo con la base de datos y la persistencia.

Las clases codificadas fueron PersonaController, ServicePersona y Persona.

4.3.4. Consulta y reporte de bitácora por persona.

Para la codificación de este caso de uso, se empleó el *framework Java Persistence API* para el mapeo con la base de datos y la persistencia.

Las clases codificadas fueron PersonaFormController, ServicePersona, Persona e Irfid. Se implementó el procedimiento JPQL:

```
	public List<Irfid> filterByDateRangePersona(Date fechaDesde, Date
fechaHasta, Persona persona){
		try {
			Timestamp pFechaDesde = new
Timestamp(fechaDesde.getTime());
			Timestamp pFechaHasta = new
Timestamp(fechaHasta.getTime());
```

```
                    Query query = em.createQuery("SELECT t FROM Irfid t
WHERE t.dateTime >= :pFechaDesde AND t.dateTime <= :pFechaHasta AND
t.tagid = :pTagId");
                    query.setParameter("pFechaDesde", pFechaDesde);
                    query.setParameter("pFechaHasta", pFechaHasta);
                    query.setParameter("pTagId", persona.getTagid());
                    returnquery.getResultList();
                } catch (NoResultException ex) {
                    returnnew ArrayList<Irfid>();
                }catch (Exception e) {
                    LOG.error(e);
                    returnnew ArrayList<Irfid>();
                }
            }
```

4.3.5. Reporte de control de asistencia general

Para la codificación de este caso de uso, se empleó el *framework Java Persistence API* para el mapeo con la base de datos y la persistencia.

Las clases codificadas fueron *ReporteController*, ServicePersona, Persona e Irfid. Se implementó el procedimiento JPQL:

```
    public List<Object[]> filterByDateRange(Timestamp fechaDesde, Timestamp
    fechaHasta){
        try {
                Timestamp pFechaDesde = new Timestamp(fechaDesde.getTime());
                Timestamp pFechaHasta = new Timestamp(fechaHasta.getTime());

        Query query = em.createQuery("SELECT p,t FROM Irfid t, Persona p
        WHERE t.dateTime >= :pFechaDesde AND t.dateTime <= :pFechaHasta
        AND p.tagid = t.tagid");
                query.setParameter("pFechaDesde", pFechaDesde);
                query.setParameter("pFechaHasta", pFechaHasta);

                returnquery.getResultList();
        } catch (NoResultException ex) {
                returnnew ArrayList<Object[]>();
        }catch (Exception e) {
                LOG.error(e);
                returnnew ArrayList<Object[]>();
        }
}
```

4.4 Pruebas

4.4.1 Pruebas de integración

Durante las pruebas de registro de los ingresos y las salidas nos encontramos con el inconveniente de que la base de datos lanzaba un error porque la conexión de la clase *DBManager* no se cerraba oportunamente.

Se realizó la corrección del defecto de límite de conexiones excedido, cerrando la conexión cada vez que se ejecuta el método *insert()* en la clase *DBManager*.

4.5 Gestión de Cambios

Durante esta etapa se realizaron cambios y adiciones en las *Interface*s gráficas de calidad de software:

- El control *spinner* en los campos de horas, minutos, segundos y milisegundos.
- El control *password* en los campos de contraseña de la base de datos y contraseña del lector
- El control de la conexión a la base de datos antes de guardar los cambios en el archivo de configuración.
- Indicador y control de campos requeridos.
- Captura de la *tag* mediante el lector RFID en el mantenimiento de personas.

5. Conclusiones

Luego de realizar este trabajo de graduación podemos concluir que los objetivos planteados en el diseño de tesis se han cumplido en su totalidad. Durante el desarrollo de la tesis, algunas de las herramientas utilizadas no fueron lo suficientemente eficientes para éste trabajo, como la herramienta para el desarrollo de *Interface*s, llamada *Pencil*, ya que, carece de una gama amplia de componentes y estilos de pantalla.

El *framework Java Persistence API* utilizado no es el óptimo para la persistencia del *DBManager*, puesto que, es muy rígido y tiene problemas para reconfiguración en caliente, de modo que se tiene que reiniciar el servidor para poder cambiar las configuraciones o el driver de la base de datos. Por esa razón, se utilizó JDBC y su driver correspondiente para la persistencia del *DBManager* ya que, es más flexible para este tipo de programación.

Con las herramientas antes mencionadas se logró tiempos de desarrollo mucho más eficientes que si no se lo hubiera hecho con ellas.

6. Recomendaciones

Recomendamos utilizar esta aplicación para futuros desarrollos en la captura de datos en tiempo real ya que permite hacerlo sin la necesidad de programar.

Una mejora a esta aplicación sería integrar una interfaz para la carga automática de drivers de otros fabricantes.

No se recomienda utilizar este tipo de tecnología en sistemas en donde se ponga en riesgo datos personales ya que, pueden ser leídos por cualquier Lector en un rango cercano.

7. Bibliografía

Referencias Electrónicas:

- *AIMglobal Asociacion de Tecnologias de Identificacion Automatica y Captura de Datos*
- *AITEX*. (01 de 10 de 2006). Recuperado el 02 de 11 de 2012, de http://rfid.aitex.es/info_rfid/frecuencias.php
- *AntenasRFID*. (01 de 10 de 2004). Recuperado el 01 de 10 de 2012, de http://www.lectoresrfid.com/Lectores_RFID/Antenas_RFID.html
- *Dipole*. (09 de 02 de 2005). Recuperado el 09 de 10 de 2012, de http://www.lectoresrfid.com
- *EPC*. (01 de 12 de 1992). Recuperado el 2012 de 09 de 16, de GS1: http://gs1ec.org
- *EPCglobalinc*. (01 de 01 de 2009). Recuperado el 01 de 10 de 2012, de http://www.epcglobalinc.org
- *EstandarEPC*. (01 de 08 de 2006). Recuperado el 02 de 10 de 2012, de http://epcglobalsp.org/standars/
- *GS1 EPC*. (01 de 05 de 1992). Recuperado el 01 de 10 de 2012, de http://gs1ec.org/contenido/index.php?option=com_content&view=article&id=50&Itemid=55
- GS1. (01 de 05 de 1992). *GS1ec*. Recuperado el 01 de 10 de 2012, de http://gs1ec.org/contenido/index.php?option=com_content&view=article&id=48&Itemid=53
- Historia RFID. (12 de 12 de 2005). *RFIDMagazine* , 23.
- *Invenligent*. (01 de 09 de 2007). Recuperado el 10 de 08 de 2012, de http://www.invenligent.com/site/library/abc_RFID.pdf
- *LectoresRFID*. (01 de 10 de 2006). Recuperado el 10 de 09 de 2012, de http://www.lectoresrfid.com/Lectores_RFID/Lectores_RFID_Clasificacion.html

- *MiddlewareRFID*. (01 de 09 de 2005). Recuperado el 01 de 09 de 2012, de http://catarina.udlap.mx/u_dl_a/tales/documentos/lep/urbina_r_rd/capitulo3.pdf
- *Nodus*. (12 de 08 de 1999). Recuperado el 10 de 02 de 2012, de http://www.sallerul.edu/ctraining/icil/RFIDNodus.pdf
- *RFID Pasaporte Electrónico*. (08 de 10 de 2005). Recuperado el 01 de 10 de 2012, de http://anita315.blogspot.com/2005/10/historia-y-tipos-de-rfid.html
- *RFIDMagazine*. (01 de 08 de 2011). Recuperado el 01 de 10 de 2012, de http://www.mas-rfid-solutions.com/docs/RFID_introduccion.pdf
- *RFIDpoint*. (03 de 10 de 2005). Recuperado el 02 de 10 de 2012, de http://www.rfidpoint.com/fundamentos/el-estandar-epc
- *RFIDSolutions*. (01 de 01 de 2002). Recuperado el 10 de 10 de 2012, de http://www.mas-rfid-solutions.com/docs/RFID_introduccion.pdf
- *RFIDsolutions*. (02 de 10 de 2008). Recuperado el 12 de 10 de 2012, de http://www.mas-rfid-solutions.com/docs/RFID_introduccion.pdf

8. Anexos

Anexo 1. Biblioteca Java de Clases Alien

Packages	
com.alien.enterpriseRFID.discovery	Para utilizar y controlar un lector, primero debe conocerse la dirección de la red, el número de puerto serie del host.
com.alien.enterpriseRFID.externalio	
com.alien.enterpriseRFID.notify	Las clases notify trabajan conjuntamente con un lector que se ejecutan de modo autónomo.
com.alien.enterpriseRFID.Reader	Las clases *Reader* son las principales para la comunicación con un lector o bien de la red o puerto serie.
com.alien.enterpriseRFID.tags	Las *Tags* desempeñan un papel muy importante en el sistema de etiquetas y lectores RFID.
com.alien.enterpriseRFID.util	Las clases contenidas en el paquete com.alien.enterpriseRFID.util proporcionan rutinas útiles para trabajar con etiquetas y lectores.

Package com.alien.enterpriseRFID.discovery

Descripción

Para utilizar y controlar un lector o *Reader*, primero debe conocerse la dirección de la red.

Tanto las *Discovery Classes* (serial y red) devuelven lectores descubiertos como *DiscoveryItem*s. Esta clase contiene un número de puntos de información claves que permiten a cualquier sistema de software identificar y ponerse en contacto con los lectores en cuestión. Con información como el nombre del lector, el tipo y la dirección.

Interface	
DiscoveryListener	DiscoveryListener es una Interfaz que implementa todos los objetos que desean recibir mensajes sobre los lectores que aparecen y desaparecen de una red.

Class	
DiscoveryItem	Detalles de un lector individual que es descubierto en un puerto serial o en la red.
NetworkDiscoveryListenerService	Es el servicio que se ocupa de los Alien *Readers* que están apareciendo y desapareciendo en la red.
SerialDiscoveryListenerService	Comprueba cada puerto serie para un *Reader*.

Exception	
AlienDiscoveryException	
AlienDiscoverySerialException	
AlienDiscoverySocketException	
AlienDiscoveryUnknownReaderException	

Package com.alien.enterpriseRFID.externalio

Class	
ExternalIO	ExternalIO *Object* representa la información acerca del evento I/O (input/output) en el lector.
ExternalIOUtil	ExternalIOUtil proporciona métodos para analizar XML y texto basado en IOLists del lector en objetos ExternalIO y vectores ExternalIOs.

Package com.alien.enterpriseRFID.notify

Descripción

Las notify classes trabajan en conjunto con un lector que se ejecuta en modo autónomo. En este modo, el lector está configurado para leer etiquetas una y otra vez sin necesidad de interacción humana. Si las etiquetas son encontradas (o eliminadas), el lector puede configurarse para enviar mensajes a los servicios listening en la red. Las notify classes implementan estos servicios listening, constantemente esperando y escuchando la notificación de mensajes de los lectores y convertirlos en objetos Java.

Interface	
MessageListener	MessageListener es una *Interface* que va a ser implementada por todos los *Object*s que desean recibir y decodificar mensajes de notificación de los lectores de funcionamiento en Modo Autónomo.

Class	
ErrorMessage	ErrorMessage toma el lugar de un Message o Mensaje, cuando ha habido algún problema descifrando la notificación del mensaje del lector.
Message	Message encapsula la información contenida en la notificación de los mensajes del lector.
MessageListenerService	MessageListenerService es una clase para recibir mensajes de notificación desde un *Reader*.

Package com.alien.enterpriseRFID.Reader

Descripción

Las *Reader classes* son lo principal para la comunicación con un lector sea en la red o el puerto serial.

Normalmente el objeto *Reader* se obtendrá desde el objeto Discovery Item. Sin embargo, si se conoce la localización del puerto serial o de la dirección de red, un objeto *Reader* puede ser instanciado directamente sin la necesidad de ninguna clase

discovery. Una vez que un objeto *Reader* válido está disponible, este ofrece al usuario una serie de comandos simples que implementa el resto de comandos descritos anteriormente.

Class	
AbstractReader	Proporciona una funcionalidad básica para una genérica *Reader* class, el manejo de comunicaciones serial y de red, y métodos de bajo nivel para el envío de comandos a los *Readers* y recepción de respuestas.
AlienClass01Reader	Esta clase permite al host interactuar con los Alien *Readers* que usan el protocol RQL (*Reader* Query Language).
AlienClass1Reader	Esta clase permite al sistema interactuar con un Alien Class 1 *Reader*.
AlienClassBPTReader	Esta clase permite al sistema interactuar con un Alien Class BPT (*Battery Powered Tag*) *Reader*.
AlienClassOEMReader	Esta clase permite al host interactuar con Alien *Readers* que usan Alien Binary *Reader* Protocol, conocido como el DLE (Data Link Escape) protocol.
AlienDLEObject	El AlienDLE*Object* class encapsula los comandos y los buffers de respuesta usados por AlienClassOEM*Reader* class para permitir al host la comunicación con los Alien *Readers* que usan el Alien Binary *Reader* Protocol.
TagMemory	
TimedSocket	Esta clase ofrece una función de tiempo de espera en la conexión del socket.

Exception	
AlienReaderCommandErrorException	
AlienReaderConnectionException	
AlienReaderConnectionRefusedException	

AlienReaderException	
AlienReaderInvalidArgumentException	
AlienReaderNoTagException	
AlienReaderNotValidException	
AlienReaderTimeoutException	

Package com.alien.enterpriseRFID.tags

Descripción

Etiquetas desempeñan un papel muy importante en el sistema *Reader* y *tag* RFID. Por esta razón hay una sola clase dedicada a almacenar y manipular información de las *tags*: la *Tag* class.

Interface	
***TagTable*Listener**	*TagTable*Listener es una *Interface* para ser implementada por todos los objetos que desean recibir notificaciones de un *TagTable* cuando la lista de *tags* cambia.

Class	
Tag	*TagObject* representa información sobre una sola *Tag*.
TagTable	*TagTable* almacena *tags* que aparecen y son quitadas fuera de fecha.
TagUtil	*Tag*Utils proporciona métodos útiles para el análisis de XML y texto basado en *tag*list desde el *Reader* a las *Tags* y arrays de *Tags*.

Package com.alien.enterpriseRFID.util

Descripción

Las clases contenidas en com.alien.enterpriseRFID.util package proporcionan rutinas útiles para trabajar con *tags* y *Readers*.

Ejemplos:

- Métodos para manipular bit a bit los datos.
- Convertir métodos y utilidades en general de I/O
- La *Interface* a las rutinas en serie javax.comm
- Clases para analizar documentos XML en tablas Hash
- Clases para generar documentos XML desde tablas Hash

Class	
API	
BitMath	Bit Math contiene un número de métodos bit a bit manipulados comúnmente por un dato byte encontrado en escenarios RFID.
Converters	Varios métodos converter y utilitarios en general para input I/O.
SerialManager	SerialManager en una *Interface* a las rutinas Serial javax.comm.
XMLReader	Se trata de un simple XML *Reader* que lee archivos XML y analiza su contenido en una table Hash.
XMLWriter	Esto es una XML utility class que escribe el contenido de una tabla Hash dentro de un texto con formato XML.

Anexo 2. Manual de Usuario

A continuación se describe el manejo de la aplicación RFID:

Para ingresar a la aplicación se debe insertar la URL:

http:// [dirección servidor]:8080/irfid

Registro de Usuarios

La pantalla de registro de usuarios es la primera que se abrirá, sirve como identificador para los usuarios que deseen ingresar al sistema, para esto cada usuario deberá estar registrado y obtendrá un Usuario y Contraseña personal.

Sesión Iniciada

En esta pantalla tenemos las siguientes opciones:

- Servidor RFID
- Configuración
- Monitor de Errores

- Mantenimiento de Personas
- Mantenimiento de Usuarios
- Reporte de control de asistencia
- Consulta de asistencia por persona

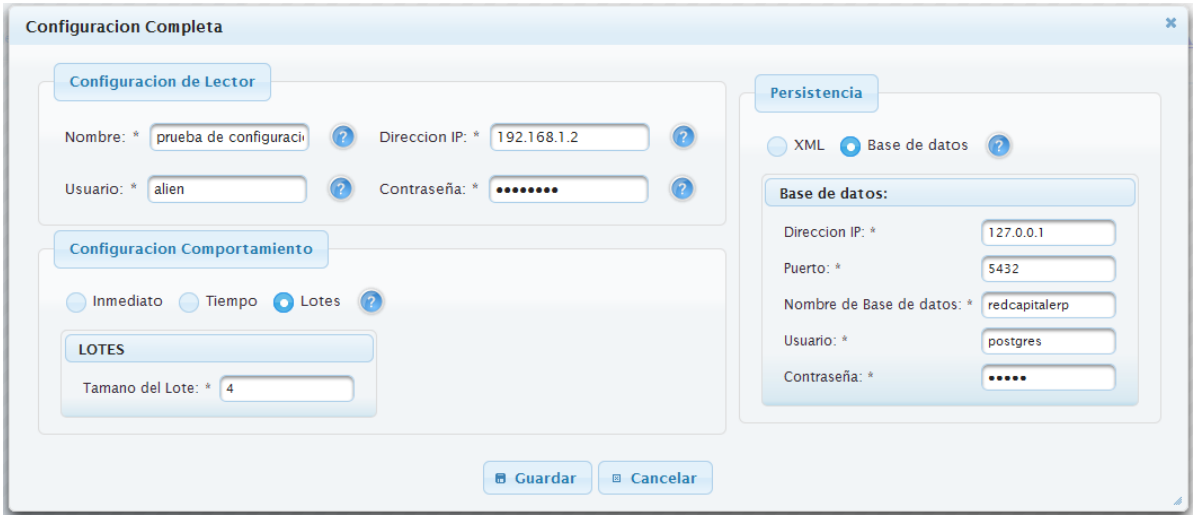
❖ Servidor RFID

Cuando se inicia la sesión, el servidor RFID se encuentra en Estado = Parado, para dar inicio al servidor se hace clic en el botón de Iniciar/Reiniciar, e indica Estado= Iniciado, en caso de que no se presente ningún error.

Se puede llegar a reiniciar el servidor, cuando algún proceso paró al servidor automáticamente, como un Error Fatal por ejemplo.

❖ Configuración

Al hacer clic en Configuración nos muestra las diferentes configuraciones que se pueden realizar en la aplicación, cabe recalcar, si no se ha iniciado el servidor de forma manual, al escoger esta opción se iniciará automáticamente.



A continuación se indica cómo debe realizarse cada configuración:

- **Configuración de Lector**

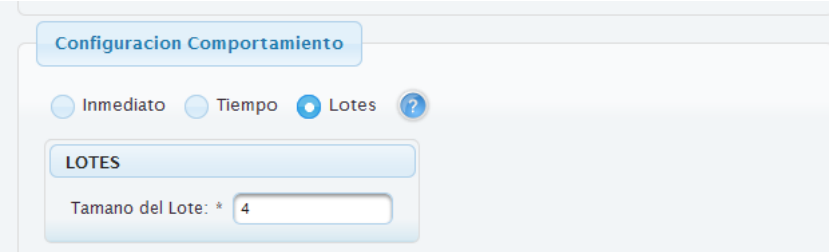
Se configuran los datos del lector,

Nombre: se ingresa el nombre con el que se autentifica al lector que se está utilizando.

Dirección IP: se ingresa la dirección IP del lector RFID con el que se va a trabajar.

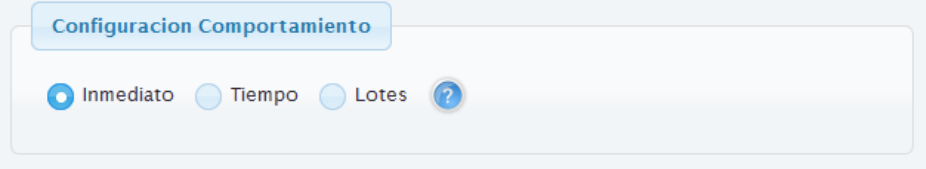
Usuario y Contraseña: el lector de la tecnología Alien usado en este proyecto nos permite ingresar un usuario y contraseña para la autentificación del lector, por default los valores son, Usuario: alien y Contraseña: password.

- **Configuración de Comportamiento:**



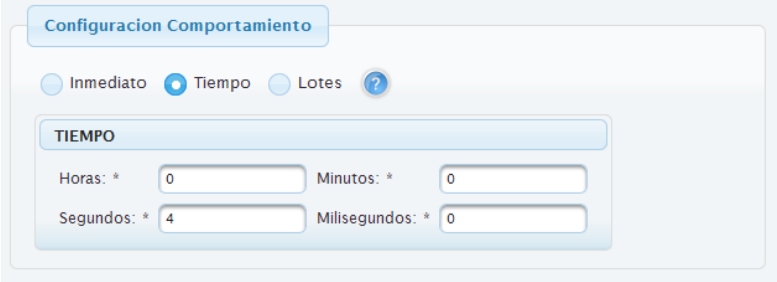
En la configuración de Comportamiento podemos indicarle al lector la manera en que el lector va a realizar la lectura de los datos de la siguiente manera:

- **Inmediato:**



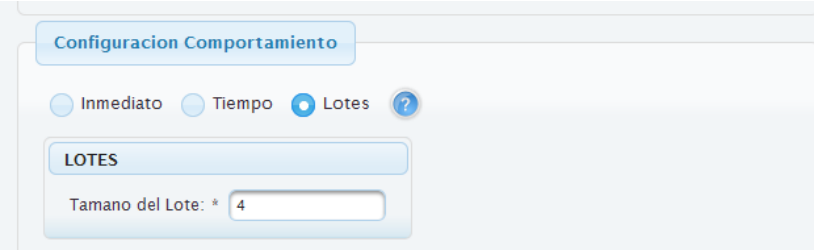
Al seleccionar ésta opción, la lectura y escritura de los datos de las *tags* que se encuentren dentro del rango de lectura se realizará de forma continua.

- **Por Tiempo:**



Esta opción permite indicar el intervalo de horas, minutos, segundos o milisegundos, en que el lector se retrasa para realizar la lectura de los datos.

- **Por Lotes:**

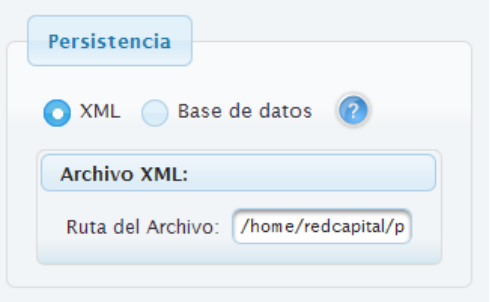


El lector realiza la lectura de los datos según el número de *tags* indicado, los datos se escriben solo si la cantidad en el buffer es mayor a la del lote ingresada, ejemplo: si se tienen 3 *tags* pero se requiere la lectura de 4, los datos no se escriben hasta que la cantidad de *tags* sean igual o mayor a 4.

- **Persistencia**

En esta configuración se decide si los datos se guardan en una Base de Datos o en un archivo XML:

- **Archivo XML**

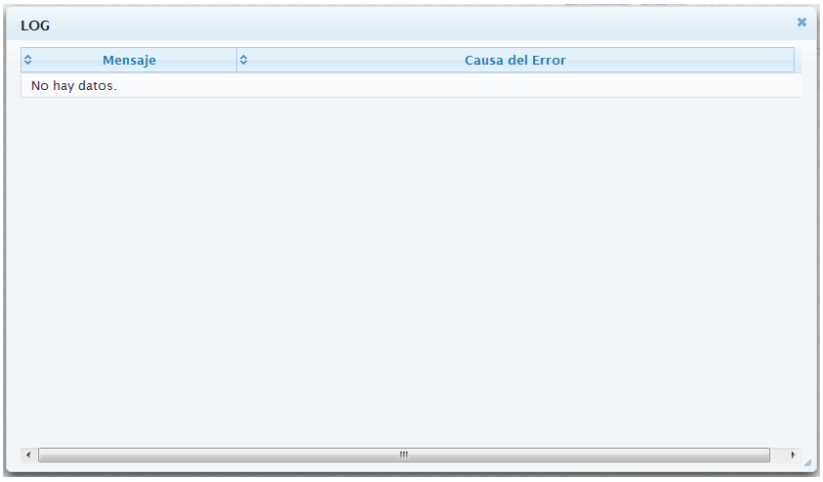


El URL que se ingrese deberá existir en el sistema de archivos y debe contar con los permisos necesarios para escritura.

- **Base de Datos**

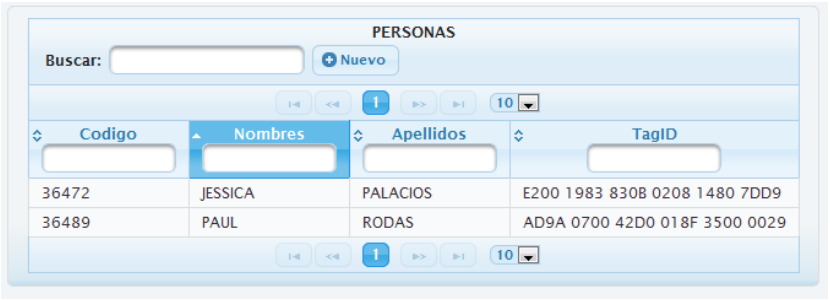
Se ingresan los datos de conexión y autenticación a la base de datos, los datos serán validados antes de guardar los cambios.

❖ **Monitor de Errores**



Aquí se registran todos los errores que ocurran durante el uso de la aplicación, indicando en nombre del error y la causa por la que se produjo.

❖ **Mantenimiento de Personas**



En esta pantalla se puede realizar una búsqueda por código, nombres, apellidos y *Tag*ID. Además, se puede realizar el mantenimiento completo de personas como agregar uno nuevo al pulsar el botón Nuevo, y para Modificar los datos de la persona se hace doble clic en los datos de la persona para la siguiente pantalla:

El botón de Editar permite modificar los datos de la persona, y para asignarle a una nueva *tag* se hace clic en el botón que está a la derecha con una lupa por ícono:

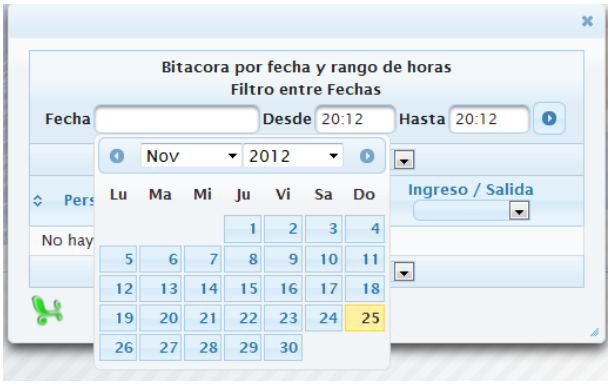
Al pulsar el botón Detectar etiquetas, permite la visualización de todas las *tags* que el Lector está leyendo en ese momento, si seleccionamos una etiqueta ya asignada nos mostrará el error caso contrario nos permitirá asignar la *tag* esa persona.

- **Mantenimiento de Usuarios**

En esta pantalla se puede realizar una búsqueda por nombre, apellido, alias, también ordenar por roles y para agregar un nuevo usuario se hace clic en el botón Nuevo:

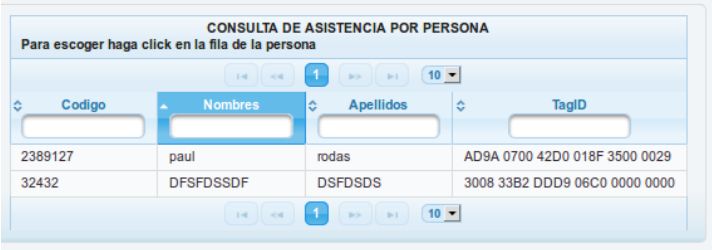
En esta pantalla ingresamos los datos del nuevo usuario, y tenemos las opciones de Guardar, Guardar y Nuevo para luego de que grabe los datos del usuario regrese a la misma, para ingresar los datos de otro usuario, y finalmente Salir.

- **Reporte de control de asistencia general.**

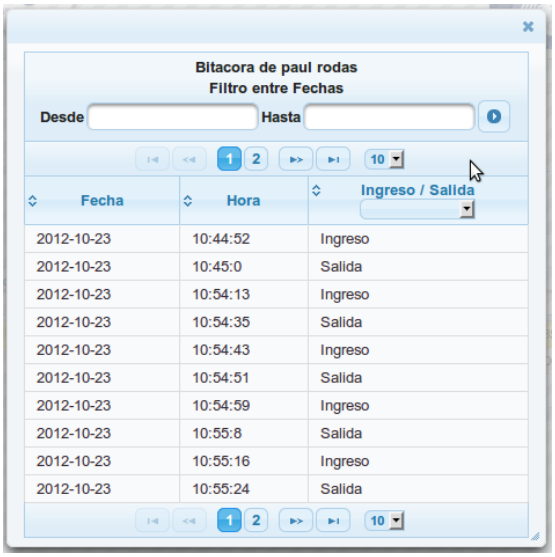


Esta pantalla nos permite crear un reporte del control de ingresos y salidas todas personas en una determinada fecha con un rango de horas establecido, y ésta información podrá ser exportada a Excel.

❖ **Consulta de asistencia por persona**



En esta pantalla se puede consultar la asistencia por persona, para esto se selecciona a la persona y se hace doble clic:



Se puede filtrar la asistencia por rango de fechas. Estos datos se pueden exportar a Excel.

DOCTOR ROMEL MACHADO CLAVIJO,

SECRETARIO DE LA FACULTAD DE CIENCIAS DE LA ADMINISTRACION

DE LA UNIVERSIDAD DEL AZUAY,

C E R T I F I C A:

Que, el H. Consejo de Facultad en sesión realizada el 26 de junio del 2012, conoció la petición formulada por la señorita **JESSICA PRISCILA PALACIOS ANDRADE** (código 35872) y señor **PAUL ADRIAN RODAS LEMA** (código 36489), que solicitan se apruebe su denuncia de tesis previa la obtención del Grado de Ingeniero de Sistemas. El tema se titula: "**DESARROLLO DE INTERFACES PARA EQUIPOS DE IDENTIFICACIÓN POR RADIO FRECUENCIA RFID APLICADO AL CONTROL DE INGRESO DE ESTUDIANTES A LOS LABORATORIOS**" . El Consejo atendiendo el informe favorable de la Junta Académica aprueba la denuncia y designa como Director del trabajo al ingeniero Pablo Esquivel León. Como Miembros del Tribunal Examinador designa a los señores profesores ingenieros Esteban Crespo y Lenin Erazo. De conformidad a las disposiciones reglamentarias los peticionarios deberán presentar su trabajo de graduación en un plazo máximo de DIECIOCHO MESES contados a partir de la fecha de aprobación, esto es hasta el 26 de diciembre de 2013.-

Cuenca, junio 29 de 2012

UNIVERSIDAD DEL AZUAY
FACULTAD DE ADMINISTRACION
SECRETARIA

UNIVERSIDAD DEL AZUAY

Cuenca, 22 de Junio 2012.

Señor Ingeniero
Oswaldo Merchán Manzano
DECANO DE LA FACULTAD DE CIENCIAS DE LA ADMINISTRACION DE LA UNIVERSIDAD DEL AZUAY.
Ciudad.

De mis consideraciones:

Por medio del presente, me permito comunicar que he procedido a revisar el Diseño de Tesis de los egresados de la Facultad, Señorita JESICA PRISCILA PALACIOS ANDRADE y el Señor PAUL ADRIAN RODAS LEMA, egresados de la escuela de Ingeniería de Sistemas, cuyo tema es "DESARROLLO DE INTERFACES PARA EQUIPOS DE IDENTIFICACION POR RADIO FRECUENCIA (RFID), APLICADO AL CONTROL DE INGRESO DE ESTUDIANTES A LOS LABORATORIOS" el mismo que cumple con todos los requisitos metodológicos y técnicos requeridos, por tal virtud no tengo ningún inconveniente en dirigir la mencionada monografía.

Por las consideraciones anotadas me permito, salvo mejor criterio, recomendar la aprobación.

Atentamente;

Ing. Pablo Esteban Esquivel León.
Docente.

UNIVERSIDAD DEL AZUAY

Cuenca, 22 de junio de 2012

Ingeniero
Oswaldo Merchán Manzano
Decano de la Facultad de Ciencias de la Administración
Ciudad

De mis consideraciones:

Nosotros, JESSICA PRISCILA PALACIOS ANDRADE con código 35872 y PAUL ADRIAN RODAS LEMA con código 36489, estudiantes de la Escuela de Ingeniería de Sistemas, solicitamos a usted de la manera más respetuosa y por su intermedio al Honorable Consejo de Facultad, se sirvan revisar el diseño de tesis titulado **"Desarrollo de interfaces para equipos de Identificación por Radio Frecuencia (RFID), aplicado al control de ingreso de estudiantes a los laboratorios."** Previa a la obtención del Título de Ingeniero en Sistemas.

Me permito sugerir el nombre del Ing. Pablo Esquivel León como director por cuanto nos ha asesorado en la elaboración del presente esquema y demás, contamos con su aprobación.

Por la favorable acogida que se sirva a la presente, suscribo de usted.

Atentamente,

Jessica Palacios A.
010415963-7

Paul Rodas L.
010525302-5

Edición autorizada de 25.000 ejemplares
Del 533.501 al 558.500
Nº 0536086

Oficio Nro. 006-2012-DIST-UDA

Cuenca, 22 de junio de 2012

Señor Ingeniero
Oswaldo Merchán Manzano
DECANO DE LA FACULTAD DE CIENCIAS DE LA ADMINSITRACIÓN
Presente.-

De nuestras consideraciones:

La Junta Académica de la Escuela de Ingeniería de Sistemas y Telemática, reunida el día 22 de junio de 2012, conoció el Proyecto de Tesis titulado "Desarrollo de interfaces para equipos de Identificación por Radio Frecuencia (RFID), aplicado al control de ingreso de estudiantes a los laboratorios. ", presentada por los estudiantes Palacios Andrade Jessica Priscila y Rodas Lema Paúl Adrian, estudiantes de la Escuela de Ingeniería de Sistemas, previo a la obtención del título de Ingenieros de Sistema.

La Junta considera que el diseño de tesis presenta una estructura teórica, metodológica y técnica objetiva y coherente, razón por la cual solicita, por su digno intermedio, el conocimiento y aprobación por parte del Consejo de Facultad.

Por lo expuesto, y de conformidad con el Reglamento de Graduación de la Facultad, recomienda designar como Director de Tesis al Ing. Pablo Esquivel , y como miembros del Tribunal al Ing. Esteban Crespo e Ing. Lenín Erazo.

Atentamente,

Ing. Marcos Orellana Cordero
DIRECTORA ESCUELA DE INGENIERIA
DE SISTEMAS Y TELEMATICA

Universidad del Azuay

Facultad de Ciencias de la Administración

Escuela de Ingeniería de Sistemas

Desarrollo de interfaces para equipos de Identificación por Radio Frecuencia (RFID), aplicado al control de ingreso de estudiantes a los laboratorios.

Diseño de Tesis previo a la obtención del Título de Ingeniero en Sistemas.

Autores: Palacios Andrade Jessica Priscila

Rodas Lema Paúl Adrian

Director: Ing. Pablo Esquivel

Cuenca, Ecuador

2012

1. Tema

Desarrollo de interfaces para equipos de Identificación por Radio Frecuencia (RFID), aplicado al control de ingreso de estudiantes a los laboratorios.

2. Antecedentes

Hoy en día, la tecnología para la identificación de objetos más usada es la del código de barras, ésta sin duda disminuye el error humano para el ingreso de datos en sistemas de información. Sin embargo, existen desventajas importantes a considerar, tales como: la escasa cantidad de datos que pueden almacenar, la imposibilidad de ser reprogramados y principalmente el método de lectura que se emplea requiere mucho tiempo, ya que se procesa uno a uno. Como consecuencia de esto, se origina la tecnología RFID: la cual pretende usar chips programables que pudieran transferir los datos que almacenaban al lector sin contacto físico, de forma equivalente a los lectores de infrarrojos utilizados para leer los códigos de barras, pero con una gran ventaja, la cual consiste en el procesamiento por lotes de información haciendo posible procesar la misma cantidad de información en una fracción muy reducida del tiempo para procesarlas con métodos tradicionales.

"En Estados Unidos se estima que las personas pasan 37 mil millones de horas al año en líneas de espera. Si este tiempo se usara de manera productiva significaría cerca de 20 millones de personas-años de trabajo útil cada año." (I. R. Mijangos n.d.)

Tomando en cuenta la cita anterior, la automatización de labores fundamentales en las instituciones es indispensable.

La utilización de RFID es un hecho cada vez más frecuente, por lo que al encontrar algunos proveedores de esta tecnología, nos vimos en la necesidad de estudiarlos para tomar una decisión correcta, reunimos algunas preguntas clave para verificar si la tecnología que nos proporcionan nos va a ser de utilidad en este proyecto, como: Pensando en la aplicación:

- ¿Qué tan cerca puede estar la etiqueta al lector?
- ¿Qué tipo de etiquetas se necesitan?
- ¿Cómo se pueden recuperar los datos almacenados?
- ¿Las APIs que nos proveen con que lenguajes de programación son compatibles?

Pensando en los lectores RFID:

- ¿Con qué frecuencia se tienen que leer las etiquetas?
- ¿Cómo se van a conectar?

Pensando en las etiquetas RFID:

- ¿A qué se les va a adherir?
- ¿Qué tamaños de etiquetas necesitamos?
- ¿Es necesario que se pueda leer y escribir en la etiqueta?

Luego de responder a éstas preguntas, sobre cuál es la tecnología que necesitamos para este proyecto, también analizamos sobre costos, ubicación, facilidades de compra y soporte.

Las compañías que analizamos fueron:

- Alien Techonology
- Intermec Technologies
- CoreRFID
- BlueBox Comunications

La falta de proveedores de esta tecnología en nuestro país y el alto costo de los equipos nos impide realizar pruebas físicas con cada uno, pero luego de analizar las características de éstas empresas proveedoras y de sus productos en base a nuestras necesidades, llegamos a la conclusión de que el candidato más viable tanto en diversidad de productos, costos bajos, facilidad de compra, transporte y la posibilidad de aprovechar la consultoría en todas de las fases de sus programas, es la compañía ALIEN, localizada en Morgan Hill, CA, dicha compañía nos brinda una

amplia gama de productos de esta tecnología, los cuales son suficientes para satisfacer las necesidades de la mayoría de empresas de nuestro medio, además de ofrecer un kit de laboratorio adecuado para el desarrollo del proyecto.

3. Selección y Delimitación del Tema

El proyecto estará centrado en la comprensión de las API's que se utilizan para la comunicación mediante redes LAN hacia los equipos de la compañía ALIEN, e igualmente en desarrollar algoritmos que efectúen la construcción automática de archivos XML y comunicación con bases de datos SQL en el momento en que lecturas efectuadas por el dispositivo registren datos.

Se desarrollarán ejercicios y aplicaciones que demuestren la tecnología, además de un manual de las API's de la marca ALIEN.

3.1 Formulación del Problema

Con la tecnología RFID intentamos reducir el tiempo de espera y mitigar la ocurrencia de errores humanos en la identificación y registro de uno o varios productos, además de proveer un medio de intercambio de datos con documentación suficiente para ser leído por software de terceros.

3.2 Sistematización del Problema

Lo que se desea, es proveer un medio para la comunicación de cualquier software con la tecnología RFID, y facilitar su implementación acortando el tiempo que ese utiliza en el estudio de las API's facilitando así el desarrollo de aplicaciones de diferente índole apoyadas por esta tecnología.

4. Objetivos

4.1 Objetivo General

Desarrollar una interfaz configurable para el intercambio de datos con otras aplicaciones utilizando las API's de la empresa ALIEN para productos RFID.

4.2 Objetivos Específicos

- Construir una interfaz para la comunicación con las API's de equipos RFID de marca ALIEN.

- Crear un manual de las diferentes API's utilizadas para el desarrollo de las aplicaciones.

- Desarrollar software utilizando API's de RFID Alien que permita la el intercambio de datos con otras aplicaciones, control de mensajes y errores, y la configuración de procesamiento de las etiquetas por lotes o por tiempos.

- Construir un aplicativo que identifique a los estudiantes que ingresan a los laboratorios de computación.

5. Justificación

La tecnología RFID (*Radio Frequency Identification*), ha tenido un crecimiento muy acelerado en los últimos años. Las posibilidades que ofrece la lectura a distancia de la información contenida solamente en una etiqueta y la capacidad de la reprogramación, nos brinda un conjunto muy extenso de aplicaciones en una gran variedad de ámbitos, desde el seguimiento de objetos o personas hasta el control de inventarios.

En este caso, el presente proyecto tiene la finalidad de demostrar el uso y beneficios de la tecnología de Identificación por Radio Frecuencia, con el desarrollo de una interfaz configurable para el intercambio de datos mediante algunos ejemplos.

6. Marco Teórico

La tecnología *RFID (Radio Frequency IDentification)*

Es un método de almacenamiento y recuperación remota de datos, basado en el empleo de etiquetas o *tags* en las que reside la información. RFID se basa en un

concepto similar al del sistema de código de barras; la principal diferencia entre ambos reside en que el segundo utiliza señales ópticas para transmitir los datos entre la etiqueta y el lector, y RFID, en cambio, emplea señales de radiofrecuencia (en diferentes bandas dependiendo del tipo de sistema, típicamente 125 KHz, 13,56 MHz, 433-860-960 MHz y 2,45 GHz). (Bermejo 2004)

Un sistema RFID consta de los siguientes tres componentes:

- **Etiqueta RFID** : compuesta por una antena, un transductor radio y un material encapsulado o chip. El propósito de la antena es permitirle al chip, transmitir la información de identificación de la etiqueta.

- **Lector de RFID**: compuesto por una antena, un transceptor y un decodificador. El lector envía periódicamente señales para ver si hay alguna etiqueta en sus inmediaciones. Cuando capta una señal de una etiqueta, extrae la información y se la pasa al subsistema de procesamiento de datos.

- **Subsistema de procesamiento de datos o *Middleware* RFID**: proporciona los medios de proceso y almacenamiento de datos.

"El RFID *Middleware* es la plataforma existente entre los lectores de tags y los sistemas de gestión empresariales para trabajar, gobernar y enviar los datos captados por el hardware RFID." Forrester Resarch

A diferencia de un *middleware* común, sus funciones básicas son la monitorización, la gestión de los datos y de los dispositivos. De hecho, extrae los datos del lector, los filtra, agrega la información y los dirige al sistema de gestión; este sistema de gestión puede ser un ERP (*Enterprise Resource Planning* o Planificación de Recursos Empresariales) o cualquier tipo de aplicación vertical (sistema de producción, almacén, etc.).

Para el intercambio de información con software de terceros necesitamos construir un archivo XML, este archivo debe contener la información que captura el lector RFID, por lo tanto, investigaremos sobre ésta tecnología, para entender la compatibilidad entre aplicaciones.

XML (*eXtensible Markup Language*). Metalenguaje extensible de etiquetas desarrollado por *World Wide Web Consortium* (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades, de ahí que se le denomina metalenguaje. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.
XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.
(W3C 2012)

Java Persistance API. Es un *framework* del lenguaje de programación Java que trabaja como un gestor de base de datos relacionales, el cual permite transportar la lógica de las operaciones IO de la base de datos a la capa de aplicación, haciendo compatible esa lógica con la mayoría de bases de datos existentes en el mercado, su estructura e instrucciones son un hibrido entre programación orientada a objetos y bases de datos relacionales, dando como resultado un nuevo lenguaje llamado JPQL, muy similar a SQL y que permite enviar las consultas a la base de datos pero independientemente de con que base se esté trabajando se utiliza los mismos comandos los cuales son traducidos por el *framework* para su posterior envió a la base de datos.

ALIEN TECHNOLOGY. Es la tecnología RFID de los equipos que utilizaremos para demostrar nuestro proyecto. La empresa ALIEN provee productos y servicios RFID a sus clientes.

Las organizaciones utilizan productos y servicios RFID de ALIEN para mejorar la eficacia, la eficiencia y la seguridad de sus cadenas de suministro, la logística y las operaciones de seguimiento de sus activos.

Entre los productos ALIEN existen *tags* RFID, lectores RFID, y servicio profesional. (ALIEN 1994).

La combinacion de estas tecnologías permite que se puedan capturar los datos mediante la tecnología RFID y se puedan enviar ya sea en formato de XML o directamente a la base de datos, mediante JPA.

7. Esquema Tentativo

CAPITULO I. Investigación de la tecnología RFID (Identificación por RadioFrecuencia)

- 1.1 Introducción RFID
- 1.2 Definición RFID
- 1.3 Seguimiento de productos con códigos EPC
- 1.4 Arquitectura RFID
- 1.5 Tipos de *Tags* RFID
- 1.6 Clasificación
- 1.7 Estandarización
- 1.8 Regulación de Frecuencias
- 1.9 Beneficios

CAPITULO II. Manual de API's de la tecnología ALIEN

- 2.1 Interfaces de reconocimiento de dispositivos en la red o conectados al puerto serial
- 2.2 Interfaces para envió y configuración de comandos RQL (*Reader Query Language*)
- 2.3 Interfaces de captura y manejo de excepciones

2.4 Interfaces de envió de mensajes hacia el dispositivo y hacia una aplicación externa

2.5 Interfaces de lectura de identificación mediante EPC (*Electronic Product Code*)

2.6 Interfaces para manejo de información y estructuras de agrupación de información de las *tag's*.

2.7 Interfaces de control de integridad de datos y convertidores

CAPITULO III. Desarrollo de software

3.1 Recopilación de requerimientos y estudio de las tecnologías a utilizar

3.2 Construcción de archivos XML para intercambio de datos de los equipos RFID marca ALIEN.

3.3 Software para el envío de información hacia una base de datos.

3.4 Consola de control de mensajes y errores.

3.5 Plataforma de configuración de procesamiento de información por lotes.

3.6 Plataforma de configuración de procesamiento de la información por tiempos.

CAPITULO IV. Aplicación de la Tecnología

4.1 Análisis

4.2 Diseño

4.3 Codificación

4.4 Integración con casos de aplicación.

4.5 Pruebas de Integración

4.6 Gestión de Cambios

8. Metodología

a) Métodos y procedimientos

- Modelado Orientado a Objetos

La técnica de modelado a objetos es un enfoque que anima a los desarrolladores de software para que trabajen y piensen en términos del dominio de la aplicación a lo largo de

la mayor parte del ciclo de vida constituidos por estructura estática, estructura dinámica y el modelo funcional. Utilizándose mayoritariamente el lenguaje unificado de modelado (UML), pensado principalmente para sistemas con gran cantidad de software.

- Modelado 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 negocio en tres componentes distintos. El patrón de llamada y retorno MVC (según CMU), se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

- Java Server Faces (JSF)

Es una tecnología *iframework* para aplicaciones java basadas en web que facilita la integración de las tecnologías HTML, CCS, JAVASCRIPT con todo el espectro de clases e interfaces java disponibles en el mercado.

- Java Persistence API (JPA)

Es un *framework* de lenguaje de programación java que maneja datos relacionales en plataformas como JAVA SE Y JAVA EE, su principal ventaja es la utilización de un lenguaje genérico llamado JPQL y teniendo que hacer configuraciones mínimas se puede integrar fácilmente a cualquier base de datos relacional

b) Técnicas

- Investigación Aplicada: Es la utilización de los conocimientos en la práctica para aplicarlos en la mayoría de los casos en provecho de la sociedad.

9. Recursos

a) Humanos

Los responsables para la elaboración y el desarrollo de la tesis serán:

RODAS LEMA PAUL ADRIAN

PALACIOS ANDRADE JESSICA PRISCILA

Asesoramiento:

ING. PABLO ESQUIVEL

b) Técnicos

Hardware:

- Laptops
- RFID labkit ALIEN (incluye *tag's* activas, *tag's* pasivas, lector, modem para comunicación de red)
- Switch WIFI
- Impresora

Software:

- PrimeFaces (Es una librería que implementa Java Server Faces de Código abierto y cuenta con un conjunto de componentes que facilitan la creación de aplicaciones web)
- Servidor de Aplicaciones JBoss (Es un servidor de aplicaciones J2EE de código abierto implementado en Java puro)
- Entorno de desarrollo Eclipse Indigo (Entorno de desarrollo de código abierto multiplataforma utilizado para desarrollo de aplicaciones mayormente JAVA)
- Enterprise Architect (software de agramado y documentación para desarrollo de proyectos)
- Java (Lenguaje de programación orientado a objetos)
- Subversion (software que se encarga de almacenar el código fuente y guardar respaldos históricos de los cambios y de los fuentes)

UNIVERSIDAD DEL AZUAY

c) Financieros

No.	Descripción	Cantidad	Costo	Total	Justificación de Gastos
1	RFID Labkit ALIEN	1	$1300.00	$1300.00	Hardware base para desarrollo
2	Resma hojas A4	1	$5.00	$5.00	Impresiones
3	CDS	5	$1.00	$5.00	Grabación Información
4	Licencia Software Enterprise Architect	1	$135.00	$135.00	Herramienta de Diseño
5	Movilización	1	$80.00	$80.00	Transporte
			TOTAL:	$1525.00	

10. Cronograma

Actividad / Duración	1 Mes				2 Mes				3 Mes				4 Mes				5 Mes				6 Mes			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
estigación tecnología RFID																								
nual de API's de ALIEN																								
sarrollo de Software																								
icación de la Tecnología																								

11. Bibliografía

Referencias bibliográficas:

- Bermejo, Ana Belén. La Identificación por Radiofrecuencia. CEDITEC-ETSIT, 2004.
- Turcu, Cristina. Development and Implementation of RFID technology. In-teh, 2009.

Referencias electrónicas:

- ALIEN, AlienTechnology, 1994, http://www.alientechnology.com. Consulta: 10 de enero de 2012.
- Mijangos, Ing. Rafael Jose Cuevas. "ITESCAM." http://www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r76473.pdf. Consulta: 27 de abril de 2012.
- W3C. *Web Extensible Markup Language.* 01 24, 2012. http://www.w3.org/XML. Consulta: 20 de febrero de 2012.

Edición autorizada de 25.000 ejemplares
Del 533.501 al 558.500
Nº 0536101