



Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Aplicaciones con Tecnología Internet

**Trabajo Especial de Grado
Elaboración de un Gem
de Ruby para Construir
un Middleware RFID
Basado en la Arquitectura
Cliente-Servidor**

Trabajo Especial de Grado
presentado ante la ilustre
Universidad Central de Venezuela
por los Bachilleres
Daniel González Torres, C.I.: 18.038.948
y
Joel Luis Ojeda Redondo, C.I.: 16.248.874
para optar por el título de
Licenciado en Computación

Tutor
Prof. Sergio Rivas

Caracas, Octubre de 2008

Acta

Quienes suscriben miembros del Jurado, designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por los **Bachilleres Daniel González Torres, C.I. 18.038.948 y Joel Luis Ojeda Redondo, C.I. 16.248.874**, con el título **Elaboración de un Gem de Ruby para Construir un Middleware RFID Basado en la Aquitectura Cliente-Servidor**, a los fines de optar al título de Licenciado en Computación, dejan constancia lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del Jurado, se fijó el 10 de noviembre de 2008 a las 8:00 a.m., para que sus autores lo defendieran en forma pública, se hizo en la Sala I de la Escuela de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, mediante una presentación oral de su contenido. Finalizada la defensa pública del Trabajo Especial de Grado, el Jurado decidió aprobarlo con una nota de puntos, en fe de lo cual se levanta la presente Acta, en Caracas a los diez días del mes de noviembre del año dos mil ocho, dejándose también constancia de que actuó como Coordinador del Jurado el profesor Sergio Rivas.

Prof. Sergio Rivas
Tutor

Prof. Andrés Castro
Jurado

Prof. Alejandro Crema
Jurado

Agradecimientos y Dedicatoria

Antes que todo, quisiera agradecer a Dios por haberme dado la oportunidad de llevar a cabo mis estudios profesionales en la carrera y la universidad que siempre quise. Quiero agradecer a mi familia por todos los consejos y el apoyo que me han brindado en todo momento. Este trabajo lo quisiera dedicar a mis padres, tías y, en especial, a mi abuela Angélica y a mi padrastro Luis, quienes han estado siempre pendientes de mis estudios y mi formación académica. Finalmente, quiero agradecer a mis compañeros y profesores por soportarme y brindarme sus consejos y ayudas.

Daniel González T.

Primeramente quisiera agradecer a toda mi familia por el apoyo brindado durante el desarrollo de mis estudios universitarios, a mis compañeros y profesores por la ayuda recibida y finalmente a la Universidad Central de Venezuela, como institución, que con sus grandes virtudes y defectos me brindó la oportunidad de culminar exitosamente mis estudios de la Licenciatura en Computación. Quisiera adicionalmente dedicar este trabajo especial de grado a la memoria de mi abuelo, Toribio Redondo Parentti quien fuera profesor jubilado de la Universidad Central de Venezuela y ejemplo de constancia, dedicación y amor a los estudios.

Joel L. Ojeda R.

Resumen

El presente Trabajo Especial de Grado trata sobre la implementación de sistemas middleware programados con el lenguaje Ruby y que emplean la tecnología RFID. La tecnología RFID provee un mecanismo versátil para la identificación, rastreo, registro y comunicación de datos relacionados a objetos de cualquier tipo. Debido a sus cualidades superiores frente a otras tecnologías de identificación, como los códigos de barras o las bandas magnéticas, RFID está reemplazando a tales tecnologías.

Este documento presenta dos partes muy importantes: el marco conceptual, en el que se abordan los aspectos relacionados a las metodologías y procesos de desarrollo, el lenguaje Ruby y la tecnología RFID, con la finalidad de exponer sus ventajas, desventajas y funcionamiento; y el marco aplicativo, en el que se exponen las actividades involucradas en el proceso de desarrollo de un gem para Ruby diseñado para implementar un middleware RFID. Un gem no es más que un paquete de software que puede ser fácilmente publicado, distribuido e instalado a través de los repositorios de RubyForge y el sistema de gestión de paquetes de Ruby (RubyGems).

El gem desarrollado se llama *rfid4r*, que significa “RFID for Ruby” dedicado a la implementación de un middleware RFID en aplicaciones Ruby. Durante el desarrollo del gem se emplearon diversas herramientas que simplificaron el desarrollo integral de cada uno de los componentes del gem. Entre las actividades más importantes del desarrollo se destaca la elaboración de una extensión de Ruby, necesaria para la utilización y manipulación de los dispositivos lectores RFID. Otra actividad a destacar es la elaboración del manejador de eventos y la interfaz de nivel de aplicación, los cuales son componentes que se agrupan bajo el concepto cliente-servidor, ya que se manejan consultas y comandos que representan peticiones que puede hacer un cliente a un servidor, y éste último le responde con el mensaje apropiado.

Palabras Clave: RFID, Ruby, Middleware, Cliente-Servidor, Gem

Contactos

Daniel González T. dangt85@gmail.com

Joel L. Ojeda R. ojeda.joel@gmail.com

Sergio Rivas sergiorivas@gmail.com

Tabla de Contenidos

Tabla de Contenidos	i
Lista de Figuras	v
I Introducción y Propuesta	1
1 Introducción	3
2 Propuesta	5
2.1 Contexto	5
2.2 Problemática	6
2.3 Objetivos Generales	6
2.4 Objetivos Específicos	6
2.5 Tecnologías	8
2.6 Plataformas	8
2.7 Procesos y Metodologías de Desarrollo	8
2.8 Alcance	8
II Marco Conceptual	11
3 Procesos y Metodologías de Desarrollo	13
3.1 Proceso de Desarrollo XP	13
3.1.1 Objetivos	14
3.1.2 Valores	14
3.1.3 Principios	15
3.1.4 Actividades	16
3.1.5 Prácticas	17
3.1.6 Aspectos Positivos	21
3.1.7 Aspectos Controversiales	22
3.2 Metodología AM	23
3.2.1 Principios	24
3.2.2 Prácticas	26
3.2.3 Aspectos Positivos	27

3.3	Resumen	28
4	Sistemas RFID	29
4.1	Componentes	29
4.1.1	Etiquetas	30
4.1.2	Lectores	32
4.1.3	Middleware	33
4.1.4	Bus de Servicios RFID	36
4.2	Aplicaciones	37
4.3	Seguridad y Privacidad	39
4.3.1	Problemas	40
4.3.2	Problemas Relacionados con la Tecnología	40
4.3.3	Privacidad y Ética	41
4.3.4	Problemas de Seguridad	42
4.3.5	Ataques	43
4.3.6	Manipulación de Radiofrecuencia	43
4.3.7	Middleware	45
4.3.8	Backend	46
4.4	Actualidad, Avances y Futuro	46
4.4.1	Tecnología	47
4.4.2	Más Etiquetas Activas	47
4.4.3	Más Manejabilidad	49
4.4.4	Más Sistemas Interconectados	50
4.5	Resumen	52
5	El Lenguaje Ruby	53
5.1	Introducción a Ruby	53
5.1.1	Metaprogramación	54
5.1.2	El Futuro de Ruby	54
5.2	RubyGems	55
5.2.1	¿Cómo Instalar RubyGems?	55
5.2.2	¿Cómo Utilizar RubyGems?	56
5.2.3	¿Cómo Crear un Gem?	56
5.2.4	Ventajas	57
5.3	Resumen	58
6	Integración de RFID con Ruby	59
6.1	¿Qué es Software Empresarial?	59
6.2	¿Qué es Integración Empresarial?	60
6.3	¿Por Qué Ruby?	60
6.3.1	Ventajas de los Lenguajes Dinámicos	60
6.4	Integración con Java	61
6.5	Extensiones para Ruby	62
6.6	Resumen	63

III	Marco Aplicativo	65
7	Adaptación de XP y AM	67
7.1	Iteraciones	67
7.2	Metas	67
7.2.1	Actividades	69
7.3	Requerimientos Generales del Sistema	69
7.3.1	Requerimientos Funcionales	70
7.3.2	Requerimientos No Funcionales	70
7.4	Aspectos Determinates del Sistema	70
7.5	Metáfora del Sistema	71
8	Desarrollo	73
8.1	Iteración 0	73
8.1.1	Planificación	73
8.1.2	Diseño	73
8.1.3	Codificación	73
8.1.4	Pruebas	75
8.2	Iteración 1	76
8.2.1	Planificación	76
8.2.2	Diseño	76
8.2.3	Codificación	76
8.2.4	Pruebas	77
8.3	Iteración 2	79
8.3.1	Planificación	79
8.3.2	Diseño	79
8.3.3	Codificación	79
8.3.4	Pruebas	81
8.4	Iteración 3	83
8.4.1	Planificación	83
8.4.2	Diseño	83
8.4.3	Codificación	84
8.4.4	Pruebas	86
8.5	Iteración 4	88
8.5.1	Planificación	88
8.5.2	Diseño	88
8.5.3	Codificación	88
8.5.4	Pruebas	89
8.6	Iteración 5	91
8.6.1	Planificación	91
8.6.2	Diseño	91
8.6.3	Codificación	92
8.6.4	Pruebas	93
8.7	Iteración 6	95
8.7.1	Planificación	95
8.7.2	Diseño	95
8.7.3	Codificación	96
8.7.4	Pruebas	96

8.8	Iteración 7	98
8.8.1	Planificación	98
8.8.2	Diseño	98
8.8.3	Codificación	99
8.8.4	Pruebas	101
IV	Conclusiones	103
9	Conclusiones	105
9.1	Mejoras a Futuro	106
9.2	Recomendaciones	107
9.3	Aportes de la Investigación	107
A	Dispositivos RFID	111
A.1	PhidgetRFID — 1023	111
A.1.1	Características del Producto	111
A.1.2	Ambiente de Programación	113
A.1.3	Recomendaciones para la Programación	114

Lista de Figuras

2.1	Propuesta de Extensión de Ruby para Sistemas RFID	7
3.1	Prácticas XP. Tomado de http://www.xprogramming.com	18
3.2	Agile Modeling. Tomado de http://www.agilemodeling.com	24
4.1	Componentes de un Sistema RFID. Tomado de [?]	30
4.2	Comunicación entre Etiquetas y Lectores RFID. Tomado de [?]	31
4.3	Kit RFID	32
4.4	Partes de un Lector RFID. Tomado de [?]	33
4.5	El Borde Empresarial en un Sistema RFID. Tomado de [?]	34
4.6	Componentes del Middleware RFID. Tomado de [?]	34
4.7	Red de Información RFID. Tomado de [?]	37
4.8	Relaciones entre los tipos de Aplicaciones RFID. Tomado de [?]	39
4.9	RFDump - Cambiando Datos en las Etiquetas. Tomado de [?]	45
4.10	Un RTLS en Acción. Tomado de [?]	48
4.11	Un RTLS en Acción. Tomado de [?]	48
4.12	Una Batería Impresa en Papel. Tomado de [?]	49
4.13	Red de malla inalámbrica. Tomado de [?]	51
4.14	RFID facilita realidad aumentada. Tomado de [?]	52
7.1	Definiciones de responsabilidades, tareas y fechas utilizando la herramientas de Assembla para el desarrollo ágil de software	68
7.2	Arquitectura del gem de Extensión de Ruby para Sistemas RFID	71
7.3	Un Posible Escenario de Uso del gem en una Aplicación RFID	72
8.1	Clases concretas y la clase abstracta <i>RfidReader</i>	76
8.2	Estructura de la tabla <i>log</i>	79
8.3	Insumos y procesos involucrados en la generación del wrapper	84
8.4	Esquema de conexión de servidores	91
8.5	Árbol de archivos del gem	95
8.6	Estructura final del gem	98
A.1	Etiquetas de disco de 30mm. Tomado de http://www.phidgets.com	112
A.2	Etiquetas en forma de tarjetas de crédito. Tomado de http://www.phidgets.com	112
A.3	Puente con diodo en relay. Tomado de http://www.phidgets.com	113

Parte I

Introducción y Propuesta

Capítulo 1

Introducción

Actualmente la demanda de mejores soluciones y tecnologías en diversas actividades de la vida cotidiana, han sacado a relucir la tecnología RFID que no había tenido mucho impacto en años anteriores [?]. Con el avance y la mejora de los materiales y procedimientos de manufactura de dispositivos electrónicos, se ha logrado reducir el tamaño y costo de dispositivos implementados para la identificación.

También se ha logrado aumentar las capacidades en cuanto a alcance, movilidad, almacenamiento de datos, etc., lo cual hace muy rentable la implementación de sistemas RFID en tareas como etiquetado de productos, identificación de animales, control de accesos, entre otras. Este trabajo se enfoca en la tecnología RFID ya que ha mostrado ser la propuesta más exitosa y con mejor futuro para este tipo de tareas.

Por otra parte, las tareas de programación a menudo involucran actividades muy engorrosas y difíciles de llevar a cabo y coordinar. Por esta razón, se ha optado por la propuesta de Ruby, ya que es un lenguaje que ha mostrado ser muy exitoso en el desarrollo ágil de aplicaciones y por ser un lenguaje muy divertido, robusto y con mucho soporte.

Por estos motivos se hace énfasis en la elaboración de un middleware RFID que permita ser adaptado a cualquier aplicación (web o stand-alone) e implementado bajo cualquier plataforma (la plataforma que ofrece Ruby, que a su vez tiene soporte para las plataformas más comunes como Windows, Linux y Mac).

El presente documento está estructurado de la siguiente manera:

Parte I - Introducción y Propuesta: en esta parte se expone la problemática derivada de las necesidades de disponer de un mecanismo que permita implementar sistemas RFID y una propuesta para solucionar tales problemas y cubrir tales necesidades.

Parte II - Marco Conceptual: en esta parte se exponen los aspectos teóricos relacionados con los procesos, metodologías y tecnologías adoptadas para la implementación del producto, con la finalidad de presentar, desde una perspectiva imparcial, tanto los aspectos positivos o ventajas, como los aspectos controversiales y desventajas de cada propuesta.

Parte III - Marco Aplicativo: en esta parte se plantean las actividades de desarrollo enmarcadas en los procesos y metodologías adoptados, con la finalidad de exponer o documentar formalmente el proceso de implementación del producto propuesto.

Parte IV - Conclusiones y Recomendaciones: en esta parte se resume de manera general el desarrollo del producto propuesto, exponiendo los aspectos de desarrollo, las dificultades, el alcance logrado, etc. Por otra parte, se hacen algunas recomendaciones para extender de las funcionalidades y características del producto.

Capítulo 2

Propuesta

En este capítulo se presenta la propuesta de Trabajo Especial de Grado, la cual muestra de manera concreta los aspectos claves de integración de software para implementaciones de sistemas RFID, considerando tecnologías, metodologías y procesos pragmáticos y vanguardistas que permitan un desarrollo ágil y adaptable a distintos sistemas de información.

2.1 Contexto

Actualmente [?] existe una gran variedad de tecnologías de identificación y movilidad que brindan la posibilidad de identificar, rastrear, ubicar, registrar, guardar y comunicar información esencial de negocios, personal o de productos. Algunas de estas tecnologías son usadas con mucha frecuencia en sistemas de información para explotar al máximo sus ventajas. Así, para los fines de esta propuesta, RFID posee cualidades superiores frente a otras tecnologías como códigos de barras o bandas magnéticas.

No se puede, por ejemplo, agregar fácilmente información a un código de barras después de que esté impreso, mientras que la información que contienen algunos tipos de etiquetas RFID puede ser modificada muchas veces. Asimismo, dado que RFID elimina la necesidad de alinear los objetos al lector, los mecanismos de seguimiento son más efectivos. Simplemente funciona de manera silenciosa, permitiendo obtener datos acerca de relaciones entre objetos y su ubicación, sin la intervención abierta del usuario o el operador.

Se puede afirmar que el gran potencial de la tecnología RFID es la posibilidad de manipular datos de manera remota e inalámbrica, por medio de comunicaciones por radiofrecuencia, a corta o mediana distancia, permitiendo gestionar muchas etiquetas o tags RFID a través de los sistemas especializados.

Estos sistemas especializados pueden ir desde sencillas aplicaciones de seguimiento y monitoreo de las etiquetas, hasta completos middleware de gestión, seguimiento, ubicación, rastreo, localización, lectura, escritura, cálculos estadísticos, identificación, entre otras funcionalidades. Este tipo de software puede aprovechar, desde diversos puntos de vista, las posibilidades que ofrece RFID.

Ahora bien, un middleware necesariamente debe estar escrito en algún lenguaje de programación y para efectos de este trabajo se propone el lenguaje de programación Ruby como herramienta ideal, por sus características de programación ágil y pragmática, que facilita el entendimiento del código, es fácil de aprender y mejora la productividad del programador.

El uso de Ruby nos ofrece la posibilidad de extender las funciones básicas del lenguaje a través del desarrollo de paquetes llamados *gems*. De esta manera podemos especializar y crear nuevas funcionalidades adaptadas a las necesidades de los desarrolladores. Las ventajas que ofrece Ruby, la necesidad de integración con otros sistemas y la popularidad que ha ganado en los últimos años, ha propiciado el desarrollo de técnicas, herramientas y código para integración de otros lenguajes como Java, C o C++ con Ruby.

Así, por ejemplo, se destaca JRuby porque permite importar código Java desde código Ruby. Basta simplemente con contar con el paquete `.jar` de Java e importarlo desde Ruby y utilizar las clases que contenga ese paquete, ya que el intérprete de JRuby se encarga del resto.

2.2 Problemática

Considerando lo expuesto anteriormente se presenta el interés de explotar y aprovechar al máximo las ventajas que ofrece la tecnología RFID para la identificación y movilidad, que permite la lectura y almacenamiento de datos de manera remota y demás beneficios mostrados en capítulos anteriores. Por otro lado, tenemos el lenguaje de programación Ruby que por su facilidad de aprendizaje y desarrollo ágil, ofrece el soporte práctico para la implementación de un sistema RFID.

De esta manera se presenta la necesidad de elaborar un sistema tipo *middleware* para usar la tecnología RFID en aplicaciones Ruby. Pero, para lograr un verdadero aporte es conveniente crear un *gem* de extensión de Ruby para que futuros desarrolladores puedan hacer uso del mismo en sus proyectos que involucren a la tecnología RFID. Más aún, aprovechar las conveniencias del sistema RubyGems para hacer más práctico el uso y distribución, bajo una licencia pública, del producto de software que se pretende realizar. Además, debido a que deberá estar publicado en los repositorios de RubyForge, es posible que otros desarrolladores puedan participar en mejoras a futuro y extender sus funcionalidades.

Finalmente, como se muestra en la Figura 2.1, se plantea como solución desarrollar un *gem* para Ruby que permita adaptar e implementar las funcionalidades del API RFID, el cual está escrito en lenguaje C/C++. Por lo que será necesario utilizar extensiones para Ruby, que dependiendo de la plataforma será necesario generar los componentes respectivos (`.so`, `.dll` o `.bundle`) para lograr una integración satisfactoria.

Una vez que el *gem* esté en total funcionamiento y debido a que tiene el soporte de Ruby podrá ser utilizado para el desarrollo de diversas aplicaciones, desde las más sencillas hasta las más completas. En este caso se pretende que el *gem* contenga funcionalidades específicas para el desarrollo de un *middleware* RFID. De esta manera podrán diseñarse, desarrollarse e implementarse aplicaciones de diversa índole para sistemas RFID completos.

2.3 Objetivos Generales

Se propone el desarrollo de un *gem* de extensión de Ruby que permita el desarrollo simple y ágil de un *middleware* RFID para implementar sistemas RFID flexibles y multiplataforma. De esta manera, se desarrollará un producto que podrá ser usado y extendido por cualquier desarrollador de la comunidad de Ruby, ya que estará disponible de forma gratuita en los repositorios de RubyForge.

2.4 Objetivos Específicos

- Utilizar la propuesta del proceso de desarrollo XP y, de ser necesario, la propuesta de la metodología de diseño AM, para el desarrollo del *gem*

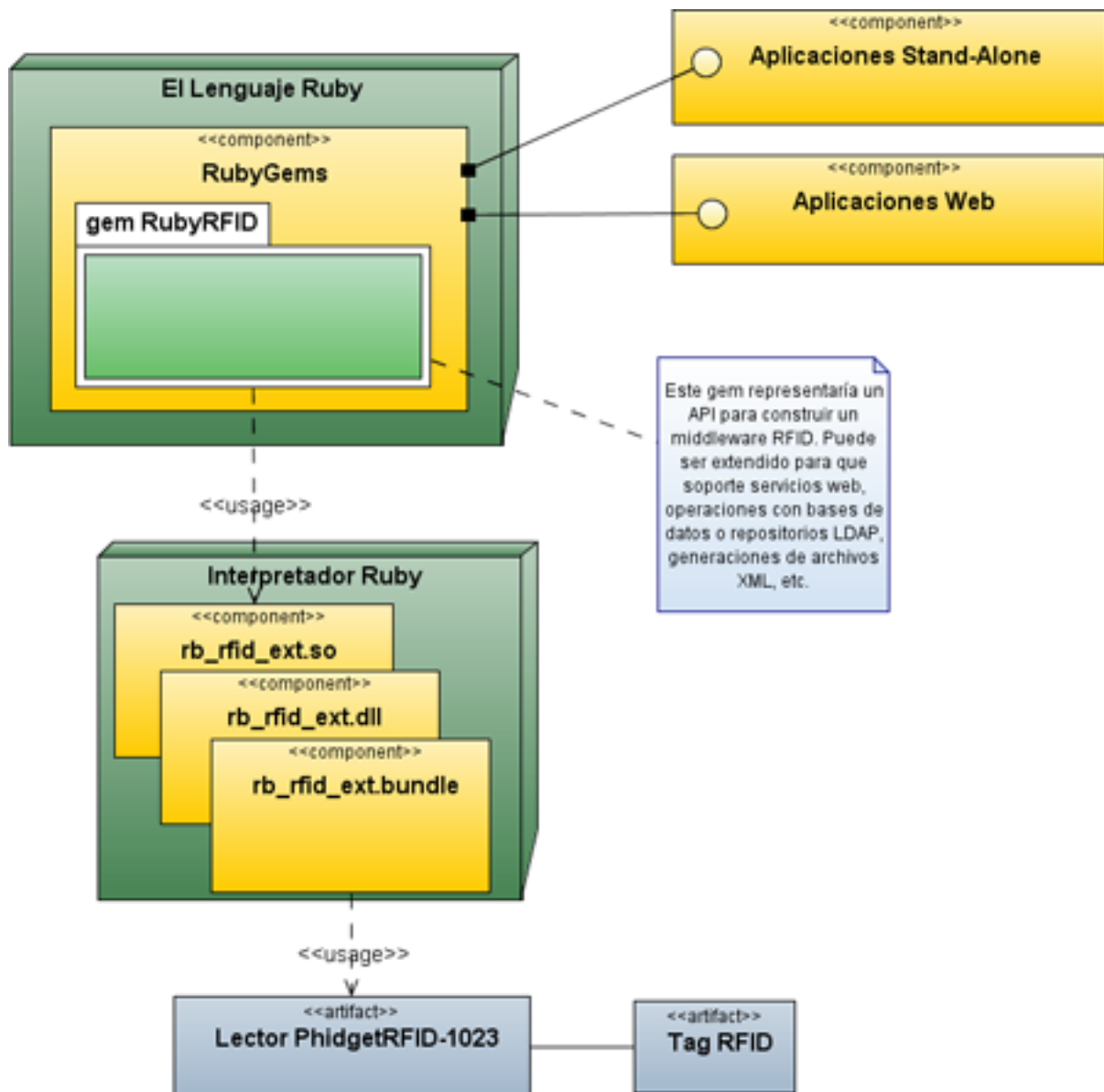


Figura 2.1: Propuesta de Extensión de Ruby para Sistemas RFID

- Implementar pruebas funcionales para comprobar el funcionamiento correcto del sistema y generar conclusiones a partir de los resultados de las mismas
- Patentar el desarrollo del **gem** bajo una licencia pública
- Publicar el **gem** en los repositorios de RubyForge
- Adaptar y generar un *wrapper* genérico del API de C/C++ del dispositivo PhidgetRFID-1023, para generar los módulos de extensión de Ruby
- Proponer la integración del API para C/C++ del dispositivo PhidgetRFID-1023 con módulos hechos en Ruby

- Proponer en la comunidad de RubyForge, la extensión del paquete para poder ser usado por la mayor cantidad de dispositivos RFID de diferentes marcas

2.5 Tecnologías

Las tecnologías propuestas contemplan muchas categorías. Sin embargo, a continuación se listarán las tecnologías que, en principio, se considerarán:

- El lenguaje de programación Ruby y el sistema de gestión de paquetes RubyGems
- El intérprete de Ruby 1.8.6 ó 1.8.7
- El compilador de interfaces SWIG 1.3.36
- El IDE de programación NetBeans 6.1 o superior
- El sistema de control de versiones Subversion 1.4.5
- El API para C/C++ del dispositivo PhidgetRFID-1023
- Los dispositivos lectores y tags de la familia PhidgetRFID

2.6 Plataformas

En principio, el desarrollo del producto se pretende realizar utilizando las tecnologías antes listadas, las cuales son independientes de la plataforma a utilizar. Ahora bien, el producto que se pretende realizar deberá contemplar soporte para cualquier tipo de plataforma, incluyendo los sistemas operativos Windows, Linux, Mac OS, las plataformas móviles, las arquitecturas de 32-bits y 64-bits, etc. En fin, deberá contemplar todo tipo de plataformas.

2.7 Procesos y Metodologías de Desarrollo

El proceso de desarrollo a utilizar será XP¹, adoptando las prácticas propuestas y complementado las prácticas de diseño y modelación con la metodología AM². La intención es producir un producto de software de la mayor calidad, utilizando una estrategia ágil y pragmática que permita ser continuada por futuros desarrolladores y que pueda ser aprovechada en la elaboración de sistemas middleware RFID, multiplataformas y fáciles de implementar.

2.8 Alcance

Los objetivos planteados para este trabajo definen un alcance particular. Este alcance está determinado por los siguientes aspectos:

- Utilizar la propuesta del proceso de desarrollo XP la metodología de diseño AM
- Desarrollar el producto utilizando la propuesta de desarrollo de paquetes de software de Ruby (RubyGems)

¹eXtreme Programming — Programación Extrema

²Agile Modeling — Modelación Ágil

- Patentar el desarrollo del gem bajo una licencia pública (código abierto)
- Publicar el gem en los repositorios de RubyForge
- Proponer la integración del API para Java o C/C++ del dispositivo PhidgetRFID-1023 con módulos hechos en Ruby

Parte II

Marco Conceptual

Capítulo 3

Procesos y Metodologías de Desarrollo

En la Ingeniería de Software en general, las metodologías de desarrollo proveen mecanismos para diseñar estrategias de desarrollo de software que promueven prácticas adoptativas en vez de predictivas, centradas en las personas o los equipos, orientadas hacia la funcionalidad y la entrega de comunicación intensiva y que requieren implicación directa del cliente. Es importante destacar que las metodologías de desarrollo describen un conjunto de métodos que pueden ser empleados en la realización de un proyecto de software.

Por otra parte, los procesos de desarrollo describen características como tareas, prácticas o funciones de los integrantes de un equipo. Se puede decir que el proceso de desarrollo es la ejecución de una o más metodologías para elaborar un producto.

En este capítulo se explican los aspectos relacionados con el proceso de desarrollo XP y la metodología AM.

3.1 Proceso de Desarrollo XP

XP se define formalmente [?] como un intento premeditado y disciplinado para definir una propuesta de desarrollo de software. Como proceso de desarrollo, XP ha sido probado con mucho éxito por más de ocho años en compañías e industrias de todos los tamaños alrededor del mundo.

Otras definiciones contemplan que XP es una metodología de la ingeniería de software, comparada y posiblemente relacionada con el desarrollo ágil de software, que describe un conjunto de prácticas que abarcan y promueven los valores XP.

XP es exitosa porque se avoca a la satisfacción del cliente. Está diseñada para llevar al cliente el software que necesita, cuando lo necesita. Permite a los desarrolladores, responder con confianza ante cambios de requerimientos del cliente, incluso en etapas avanzadas del ciclo de desarrollo. Además hace énfasis en el trabajo en equipo. Líderes, clientes y desarrolladores son todos parte del equipo dedicado a producir software de calidad. XP implementa una manera simple, pero efectiva de posibilitar el estilo de desarrollo en grupo.

XP mejora los proyectos de software al basarse en cuatro fundamentos básicos: comunicación, simplicidad, feedback y coraje¹. Los programadores XP se comunican con sus clientes y con sus compañeros programadores. Ellos mantienen sus diseños sencillos, reciben respuestas probando el

¹se refiere a la actitud que asume un programador al enfrentar cambios en los requerimientos

software desde el primer día, entregan el sistema al cliente tan rápido como puedan e implementan los cambios sugeridos. Con estos fundamentos, los programadores XP están dispuestos a responder a los cambios de requerimientos y de tecnologías. Es por esto que XP se diferencia de otros procesos de desarrollo porque hace mayor énfasis en la adaptabilidad que en la previsibilidad. En este sentido, se considera que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos.

Existen muchas pequeñas partes o tareas que no tienen sentido de manera individual, pero cuando se combinan se puede percibir el beneficio. Esto es una novedad significativa en los procesos de desarrollo tradicional de software.

3.1.1 Objetivos

El principal objetivo de XP es la reducción del costo asociado al cambio. En los métodos tradicionales de desarrollo de sistemas como, por ejemplo, SSADM² o RUP³, los requerimientos del sistema eran determinados al comienzo del desarrollo y luego acomodados a partir de ese punto. Esto significaba que el costo asociado a cambios en los requerimientos sería muy elevado.

XP contempla que, para reducir el costo al cambio, es necesario tomar en cuenta una serie de principios, prácticas y valores básicos. Al aplicar XP en un proyecto de desarrollo, éste debe ser más flexible con respecto al cambio.

Beck describe [?] XP como:

- Un intento por reconciliar a la humanidad con la productividad
- Un mecanismo para el cambio social
- Una vía para mejorar
- Un estilo de desarrollo
- Una disciplina de desarrollo de software

3.1.2 Valores

Inicialmente en 1999 se contemplaron cuatro valores. En la segunda edición del libro de Beck, *Extreme Programming Explained*, se añadió un nuevo valor. Los cinco valores son:

Comunicación: la construcción de sistemas de software requiere de mecanismos de comunicación entre los desarrolladores. En las metodologías formales de desarrollo de software, esta tarea se lleva a cabo a través de la documentación. Las técnicas XP se pueden ver como métodos para la construcción y difusión de conocimiento institucional de manera rápida entre los miembros del equipo de desarrollo. La meta es dar a los desarrolladores una vista compartida del sistema que concuerde con la vista que los usuarios tienen del sistema. Para esto, XP contempla los diseños simples, las metáforas comunes, la colaboración entre usuario y programadores, la comunicación verbal frecuente y la retroalimentación.

Simplicidad: XP promueve empezar con la solución más simple. Las funcionalidades extras pueden ser agregadas luego. La diferencia entre esta propuesta y otros métodos convencionales de desarrollo de sistemas es el enfoque en el diseño y codificación de las necesidades del día en vez

²Structured Systems Analysis and Design Method — Método de Diseño y Análisis de Sistemas Estructurados

³Rational Unified Process — Proceso Unificado Racional

de las necesidades del mañana, la próxima semana o el próximo mes. Sin embargo, esto puede significar un mayor esfuerzo en cambiar el sistema el día siguiente, pero esto es compensado por la ventaja de no tener que invertir en posibles cambios en los requerimientos futuros antes de hacerse muy relevantes. Codificar y diseñar para requerimientos futuros e inciertos implican un riesgo de inversión de recursos en algo que puede no ser necesitado. Además, un diseño simple con código sencillo puede ser fácilmente entendido, lo cual mejorará la comunicación entre el equipo de programadores.

Retroalimentación: está asociada directamente con la comunicación y la simplicidad. Las fallas en el sistema son reportadas fácilmente escribiendo tests unitarios que prueben que cierta porción de código fallará. Esta respuesta directa del sistema le dirá a los programadores que deben corregir el código. El cliente debe poder probar el sistema periódicamente de acuerdo a los requerimientos funcionales. La retroalimentación puede verse desde distintos puntos de vista:

- Desde el punto de vista del sistema, la implementación de tests unitarios o de integración permitirá a los programadores tener respuesta directa acerca del estado del sistema antes y después de implementar cambios
- Desde el punto de vista del cliente, los tests funcionales o de aceptación son escritos por los clientes y los programadores de pruebas. Ellos obtendrán respuestas concretas acerca del estado del sistema. Estas revisiones podrían planearse cada dos o tres semanas, de manera que el cliente pueda dirigir fácilmente el desarrollo
- Desde el punto de vista del equipo, cuando los clientes tienen nuevos requerimientos, el equipo da directamente una estimación del tiempo que tomará la implementación

Coraje: se refiere al planteamiento de siempre diseñar y programar para hoy y no para mañana. Este es un esfuerzo para evitar agobiarse en el diseño y la implementación de otras funcionalidades. El coraje permite a los desarrolladores sentirse cómodos con la refactorización. Esto significa la revisión del sistema existente y modificarlo para que las futuras funcionalidades puedan ser incorporadas fácilmente. Otro escenario que requiere coraje es decidir deshacerse de código que puede ser obsoleto, sin importar el esfuerzo que requirió el ser creado. También, coraje significa persistencia: un programador puede estar estancado en un problema un día completo, y luego resolver el problema el día siguiente, sólo si el programador es persistente.

Respeto: este valor se puede expresar de diferentes maneras. En XP, los miembros del equipo expresan respeto mutuo porque los programadores no deben hacer cambios que dañen la compilación, que hagan que pruebas unitarias fallen, o que de alguna manera retrasen el trabajo. Los miembros del equipo respetan su trabajo al buscar siempre la mayor calidad y el mejor diseño para una solución a través de la refactorización. También, la adopción de los cuatro valores anteriores conllevan a ganarse el respeto de los demás miembros del equipo. Nadie en el equipo debe sentirse menospreciado o ignorado. Esto asegura un alto nivel de motivación y promueve la lealtad al equipo y a las metas del proyecto. Este valor es muy dependiente de los otros valores y está orientado a las personas en el equipo.

3.1.3 Principios

Los principios que forman la base de XP se fundamentan en los valores antes descritos e intentan fomentar las decisiones en los proyectos de desarrollo de sistemas [?]. Estos principios tienen la finalidad de extender las características de los valores para ser traducidos más fácilmente en guías concretas para situaciones prácticas.

1. El feedback es más útil si se hace rápidamente. El tiempo entre una acción y su respuesta es un aspecto crítico para entender y hacer los cambios. En XP, a diferencia de los métodos tradicionales de desarrollo, el contacto con el cliente ocurre cada cierto intervalo corto de tiempo. El cliente así tiene una idea clara del sistema que está siendo desarrollado y puede dar un feedback para acelerar el desarrollo según se necesite.
2. Los tests unitarios también contribuyen al principio de la rápida retroalimentación. Cuando se escribe código, los tests unitarios proveen una retroalimentación directa sobre cómo el sistema reacciona ante los cambios que se hacen. Si, por ejemplo, los cambios afectan a una parte del sistema que no está al alcance del programador que la hizo, ese programador no notará el cambio. Existe una alta posibilidad de que se detecte un error cuando el sistema está en producción.
3. Asumir simplicidad se refiere a tratar los problemas como si su solución fuese extremadamente simple. Los métodos tradicionales de desarrollo de sistemas contemplan que se debe planear en base al futuro y codificar para permitir la reusabilidad. XP no acepta estas ideas. Según la metodología XP el hacer grandes cambios de una vez no funciona. XP aplica cambios incrementalmente, por ejemplo, un sistema que deberá tener pequeñas entregas cada tres semanas. Aplicando de esta manera muchos pasos pequeños se tendrá mayor control sobre el proceso de desarrollo y el sistema que se está desarrollando.
4. El principio de contemplar cambios se refiere a adoptar que existe la posibilidad de presentarse cambios. Por ejemplo, si en una de las reuniones los requerimientos del cliente parecen cambiar dramáticamente, los programadores deben adoptar estos cambios y planificar nuevos requerimientos para la próxima iteración.

3.1.4 Actividades

XP describe cuatro actividades básicas que se llevan a cabo dentro del proceso de desarrollo de software [?].

Codificar: los defensores de XP argumentan que el único producto verdaderamente importante en el proceso de desarrollo de sistemas es el código. Sin el código no se tiene nada. Codificar puede consistir en dibujar diagramas que generarán código, teclear sentencias de código para un sistema web o un programa que necesite ser compilado. Codificar puede enfrentar varias alternativas para un problema de programación. Se pueden codificar todas las soluciones y determinar, a través de tests automatizados, cuál es la que mejor se adapta. Codificar también puede ayudar a comunicar ideas acerca de problemas de programación. Un programador que esté enfrentando un problema complejo y difícil de explicar su solución a los demás programadores, puede codificarlo y usarlo para demostrar lo que significa. El código siempre es claro y conciso y sólo puede ser interpretado de una sola manera.

Probar: no se puede estar seguro de algo a menos que se haya probado ese algo. El desarrollo de tests no es una necesidad directa del cliente. Muchos productos de software son entregados sin las pruebas adecuadas y aún así funcionan. En el desarrollo de software, XP establece que no se puede estar seguro que una función sirva a menos que se pruebe. Esto trae como consecuencia que:

1. No se puede estar seguro de que lo que se codificó es lo que se quiere expresar. Para probar esto, XP usa tests unitarios. Estos son tests prueban código de manera automática. El programador trata de escribir la mayor cantidad de tests para hacer fallar el código que

está escribiendo. Si los tests arrojan resultados positivos, entonces se dice que el código pasó las pruebas.

2. No se puede estar seguro de que lo que se expresó es lo que se debió haber expresado en el código. Para probar esto, XP usa tests de aceptación basados en los requerimientos del cliente durante la fase de exploración.

Comunicar: los programadores no necesariamente saben algo acerca del aspecto de negocios del sistema que se está desarrollando. La función del sistema se determina por éstos aspectos de negocio. Para que los programadores entiendan cuál debe ser la funcionalidad del sistema, deben estar dispuestos a escuchar a clientes, posiblemente indagar en las necesidades del cliente, tratar de entender el problema de negocios y proveerle al cliente cierto feedback para que el mismo cliente pueda mejorar su entendimiento del problema.

Diseñar: se puede decir que el desarrollo de un sistema no necesita más que codificación, pruebas y escuchar. Si esas actividades se llevan a cabo bien, los resultados serán siempre un sistema que funcione. En la práctica esto no funciona. Es posible pasar un tiempo sin diseñar, pero es inevitable quedarse atascado en cierto momento. El sistema se vuelve muy complejo y las dependencias dentro del sistema dejan de ser claras. Esto se puede evitar creando una estructura de diseño que permita organizar la lógica del sistema. Los buenos diseños permiten evitar muchas dependencias dentro del sistema. Es decir, que cambiando una parte del sistema no se afecten otras partes del mismo sistema.

3.1.5 Prácticas

En XP [?], cada miembro activo del proyecto es parte integral del equipo completo. Los equipos se forman a partir del representante de negocios, conocido como el cliente, el cual se reúne con el equipo y trabaja con él a diario.

Los equipos XP usan una forma simple de planificación y seguimiento para decidir qué se debe hacer a continuación y predecir cuándo un proyecto estará listo. Al enfocarse en el valor del negocio, el equipo produce el software en pequeñas entregas bien integradas que pasan todos los tests que el cliente define.

Los programadores XP trabajan en parejas y como un grupo, con diseños simples y código probado exhaustivamente, mejorando continuamente el diseño para mantenerlo adecuado para las necesidades actuales.

Los equipos XP mantienen el sistema integrado y en ejecución en todo momento. Los programadores escriben todo el código que irá a producción en parejas, y todos trabajan juntos en todo momento. Ellos codifican de manera consistente de manera que todos puedan entender y mejorar todo el código según sea necesario.

Los equipos XP comparten una visión simple y común de cómo el sistema debe lucir. Todos trabajan en un lugar que puede ser sostenible indefinidamente.

A lo largo de la vida de todo proyecto XP se desarrollan diferentes actividades y tareas, las cuales deben ser llevadas a cabo siguiendo las mejores prácticas para lograr los mejores resultados. Las principales actividades que se llevan a cabo en proyectos XP se describen a continuación:

Equipo Completo

Todo equipo de desarrollo XP debe incluir un representante de negocios (el cliente) el cual provee los requerimientos, establece las prioridades y dirige el proyecto. Es preferible si el cliente o al menos uno



Figura 3.1: Prácticas XP. Tomado de <http://www.xprogramming.com>

de sus ayudantes es un usuario final que conoce del dominio y lo que es necesario. El equipo debe también incluir programadores. Puede incluir testers⁴, los cuales asisten al cliente para definir tests de aceptación. Los analistas sirven como ayudantes del cliente para definir los requerimientos. Comúnmente hay un coach, el cuál ayuda a mantener el equipo activo y facilita el proceso de desarrollo. Debe haber un manager, el cual debe, entre otras actividades, proveer los recursos, manejar la comunicación externa y coordinar las actividades del equipo. No necesariamente estos roles pueden llevarse a cabo exclusivamente por sólo un individuo: todos los miembros de un equipo XP contribuyen de alguna manera. Los mejores equipos no tienen especialistas, sino personas que contribuyen de manera general y que tienen habilidades especiales.

Planificación

XP sugiere dos aspectos claves para la planificación del desarrollo de software: predecir qué se quiere lograr para el día esperado y determinar qué hacer a continuación. El énfasis está en dirigir el proyecto, lo cual es bastante sencillo, en vez de predecir exactamente qué será necesario y cuánto tiempo tomará, lo cual es bastante difícil. Existen dos pasos claves a seguir en la planificación XP:

- La planificación de una entrega es una práctica en la que el cliente presenta las características deseadas a los programadores, y ellos estiman la dificultad. Con los estimados de los costos y con el conocimiento de la importancia de estas características, el cliente define un plan para el proyecto. Los planes iniciales para una entrega son imprecisos: ni las prioridades ni las estimaciones son sólidas, y hasta que el equipo empiece a trabajar, no se sabe qué tan rápido se irá. Sin embargo, el primer plan de una entrega es lo suficientemente preciso para tomar decisiones con lo que el equipo hace revisiones con regularidad a la planificación.
- La planificación de una iteración es la práctica con la cual el equipo recibe instrucciones cada dos semanas. Los equipos XP construyen software en iteraciones de dos semanas, entregando software funcional al final de cada iteración. Durante la planificación de una iteración, el cliente presenta las características deseadas para las siguientes dos semanas. Los programadores descomponen esas características en tareas y estiman su costo (en un nivel más detallado que en la planificación de entregas). En base a la cantidad de trabajo acumulado en la previa iteración, los equipos determinan qué será asumido para la iteración actual.

Estos pasos en la planificación son muy simples. Sin embargo, proveen muy buena información y excelente manejo para el cliente. Cada dos semanas, el progreso logrado es muy notable. El trabajo no

⁴Programadores especializados en escribir el código de los tests

se aprecia de acuerdo a un porcentaje: se logró la meta o no se logró. Este enfoque en la apreciación de resultados conlleva a una paradoja: por una parte, con mucha notoriedad, el cliente está en la posición de cancelar el proyecto si el progreso no es suficiente. Por otra parte, el progreso es tan notorio y la habilidad de decidir qué quiere a continuación es tan completa que los proyectos XP tienden a dar más de lo que se necesita con menos estrés y presión.

Pruebas del Cliente

Como parte la presentación de cada característica deseada, el cliente defina uno o más tests de aceptación para mostrar que la característica funciona. El equipo construye estos tests y los usa para probar, a sí mismos y al cliente, que la funcionalidad ha sido implementada correctamente. La automatización es importante porque en la presión del tiempo, se omiten los tests manuales.

Los mejores equipos XP tratan a los tests de los clientes de la misma manera que los tests de los programadores: una vez se ejecuta sin errores el test, el equipo trata de mantenerlo así, en adelante. Esto significa que el sistema sólo mejora al agregar funcionalidades y no al quitarlas.

Pequeñas Entregas

Los equipos XP hacen las entregas de dos maneras:

1. El equipo entrega, en cada iteración, software ejecutable y probado, que contiene el valor de negocio que el cliente eligió. El cliente puede usar este software para cualquier propósito, ya sea evaluación o para poner en producción para el usuario final (es lo más recomendado). El aspecto más importante es que el software es visible y se entrega al cliente al final de cada iteración.
2. Los equipos XP también hacen entregas a sus usuarios finales. Los proyectos Web se entregan diariamente. En proyectos internos, las entregas pueden ser mensuales o más frecuentemente.

Puede parecer imposible crear buenas versiones dentro de estos periodos de tiempo, que los equipos XP lo hacen todo el tiempo.

Diseño Simple

Los equipos XP construyen software a partir de diseños simples. Se empieza con un diseño simple y, a través de las pruebas de los programadores y las mejoras de diseño, se mantiene un esquema sencillo de desarrollo. Los equipos XP mantienen el diseño precisamente adaptado para la funcionalidad actual de sistema. De esta manera no se desperdicia trabajo y el software está siempre listo para el siguiente requerimiento.

Los diseños en XP no se hacen una sola vez, sino todo el tiempo. Hay pasos en el diseño durante la planificación de una entrega o la planificación de una iteración. Además, los equipos realizan sesiones rápidas de diseño y hacen revisiones a través de la refactorización a lo largo del proyecto completo. En un proceso incremental e iterativo como XP, los buenos diseños son esenciales. Es por eso que XP se centra mucho en el diseño a lo largo del desarrollo.

Programación en Parejas

En XP, todos los software en producción se construyen por programadores, sentados frente a la misma máquina de desarrollo. Esta práctica asegura que todo el código de producción es revisado por al menos un programador. Lo cual resulta en un mejor diseño, mejores pruebas y mejor código.

Puede parecer ineficiente tener dos programadores haciendo el trabajo de uno sólo, pero lo contrario es una certeza. Las investigaciones en la programación en parejas muestran que se produce mejor código en casi el mismo tiempo en el que se produce en la programación normal.

Algunos programadores no desean poner en práctica la programación en pareja sin haberla intentado. Requiere práctica hacerlo bien para ver los resultados. La mayoría de los programadores que aprenden a programar en parejas prefieren esta práctica.

La programación en parejas también ayuda a mejorar la comunicación en el equipo. Cuando las parejas se intercambian, se intercambian los conocimientos del otro. Los programadores aprenden, mejoran sus habilidades, se hacen más valiosos para el equipo y para la compañía.

Mejoramiento del Diseño

XP se centra en entregar un producto de valor de negocios en cada iteración. Para lograr esto a lo largo del curso del desarrollo de todo el proyecto, el software debe ser bien diseñado. La alternativa sería reducir la velocidad de desarrollo y eventualmente, llegar a un punto en el que no se puede seguir avanzando. Es por esto que XP usa un proceso de diseño continuo conocido como refactorización.

El proceso de refactorización se basa en eliminar duplicaciones (una señal segura de un diseño pobre), e incrementar la cohesión del código a la vez que se disminuye el acoplamiento. La alta cohesión y el bajo acoplamiento son señales de un código bien diseñado que han sido reconocidas por casi 30 años. El resultado es que los equipos XP empiezan con un buen diseño, simple y que se mantiene así durante todo el desarrollo del software. Esto conlleva a que el equipo mantenga o aumente la velocidad de desarrollo del proyecto.

La refactorización, es soportada por el diseño comprensivo de los tests para asegurarse de que nada se dañe mientras el diseño evoluciona. Por eso, los tests del cliente y de los programadores son un factor crítico para continuar el desarrollo. Las prácticas XP se soportan unas con otras. Son más valoradas cuando se hacen juntas que cuando se hacen por separado.

Integración Continua

Los equipos XP mantienen el sistema completamente integrado en todo momento. Las integraciones se hacen varias veces al día. El beneficio de esto se puede apreciar al comprar un proyecto XP con un proyecto en el que el proceso de integración se hace semanalmente o con menor frecuencia lo cual conlleva a problemas de integración en los que todo se daña y no se sabe por qué.

Las integraciones que se hacen con poca frecuencia conllevan a serios problemas en los proyectos de software. Primero que nada, aunque la integración es crítica para entregar código bueno y ejecutable, el equipo no está acostumbrado a eso, y a menudo se delega la responsabilidad a personas que no están familiarizadas con el proyecto. Segundo, las integraciones que se hacen con poca frecuencia tienen muchos errores a nivel de código. Los problemas se acarrea si no son detectados por los tests. Tercero, los procesos de integración pobres conllevan a un congelamiento del código. Es decir, que pasarán largos períodos de tiempo en los que los programadores pueden estar trabajando en características importantes, pero esas características deben ser aplazadas. Esto debilita la posición del equipo en el mercado o con el usuario final.

Código de Propiedad Colectiva

En un proyecto XP, cada par de programadores puede mejorar cualquier código en cualquier momento. Esto significa que todo el código recibe el beneficio de la atención de muchas personas. Lo cual incrementa la calidad del mismo y reduce sus defectos. Existe también otro beneficio importante: cuando el código es propiedad de un individuo y un programador descubre que necesita una característica

en algún lugar del código del cuál no es propietario, estas características requeridas son, a menudo, puestas en lugares equivocados. El propietario del código puede estar muy ocupado para hacerlo. Así que el programador coloca esta característica en su propio código en donde no pertenece. Esto conlleva a códigos poco elegantes, difíciles de mantener, con mucha duplicación y con baja cohesión.

La propiedad colectiva puede ser un problema si las personas que contribuyen en el código lo hacen sin entenderlo. XP evita estos problemas mediante dos técnicas claves: los tests de los programadores capturan los errores y la programación en parejas. Lo que significa que la mejor manera de trabajar con código que no es familiar es con un experto. Adicionalmente, para asegurar que se hacen bien las modificaciones cuando se necesita, esta práctica distribuye el conocimiento en todo el equipo.

Estándares de Codificación

Los equipos XP siguen un estándar de codificación, de manera que todo el código en el sistema parezca haber sido escrito por un sólo individuo. Las especificaciones de los estándares no son muy importantes. Lo que sí es importante es que todo el código parezca familiar para soportar la propiedad colectiva.

Metáforas

Los equipos XP desarrollan una visión común de cómo el sistema completo funciona y de cómo es el flujo de trabajo. Esto es lo que se conoce como metáfora. La metáfora es una simple descripción de cómo el programa debe funcionar.

Algunas veces es difícil encontrar una metáfora. En todo caso, con o sin una imagen vívida, los equipos XP usa un sistema común de nombres para asegurarse de que todos entienden cómo funciona el sistema y dónde buscar una funcionalidad o encontrar el lugar en dónde colocar una funcionalidad. Una metáfora puede ser un diagrama sencillo, un conjunto de figuras que describen el comportamiento de un módulo o cualquier otro elemento que sugiera una descripción del sistema.

Ritmo Sostenible

Los equipos XP se conforman a largo plazo. Trabajan fuerte a un ritmo que puede ser sostenido indefinidamente. Esto significa que el equipo puede trabajar tiempo extra si es efectivo, y que normalmente trabajan de tal manera que se maximiza la productividad.

3.1.6 Aspectos Positivos

La propuesta XP [?] es soportada por el reconocimiento de sus aspectos positivos. Primeramente, los tests unitarios en el código es una práctica generalmente alentada y reconocida como un factor clave para obtener un software de alta calidad. Si a eso se le agrega la exigencia de XP de que se hagan constantemente, durante cada etapa de codificación, quizás en el peor de los casos pudiera parecer un exceso. De manera semejante, la integración continua es aceptada y recomendada para evitar catástrofes ocasionadas por defectos no detectados a tiempo.

El énfasis en la simplicidad y la refactorización es encontrado como un factor saludable en la práctica de programación, ya que normalmente se asocia el exceso de código con una lógica deficiente, un diseño innecesariamente complejo, problemas para el mantenimiento del sistema y un nicho para encontrar defectos que demeritan la calidad del producto.

También se puede ver como positivo el hecho de que, filosóficamente, XP tiene un enfoque extremadamente humano, siendo este un aspecto que el resto del campo del software debería tratar de emular. Desde el lado del programador, como ejemplos de este enfoque humano, tenemos la premisa

de la semana de 40 horas, el alto valor que se le da a la comunicación y el rol protagónico que toma el programador en la etapa de planeación.

Por el lado del cliente también se percibe el enfoque humano, ya que tenemos su presencia constante en las instalaciones del desarrollador, el diálogo que se fomenta entre el cliente y el resto del equipo de programación, la declaración de que escuchar al cliente es una de las cuatro actividades esenciales de la programación y el hecho de que se le otorga el mayor peso en la planeación.

Otro aspecto positivo que se puede resaltar de XP es la naturaleza de su curva de aprendizaje y la curva relacionada al costo del cambio. XP requiere de la aplicación de técnicas y prácticas muy sencillas que reducen el costo relacionado al cambio y requieren menor esfuerzo para aprenderlas y llevarlas cabo.

3.1.7 Aspectos Controversiales

La propuesta XP [?] ha despertado cierta controversia al rededor de su utilidad y alcance reales. Desde los inicios de XP se ha afirmado que no es la metodología que va a resolver todos los problemas en Ingeniería de Software y se han resaltado sus limitaciones, señalando aquellos ambientes o proyectos en los que no se debería intentar aplicarla. Por ejemplo:

- En proyectos donde la cultura del cliente esté demasiado orientada a metodologías tradicionales no ágiles o a largas jornadas de trabajo
- En proyectos donde se involucren equipos de más de 20 programadores
- Si no es posible disminuir la curva costo/tiempo
- Si la tecnología o el entorno no permiten realizar integraciones frecuentes o realizar pruebas continuamente
- Si la distribución física del mobiliario impide la programación en pares o si no todos los programadores se encuentran en el mismo sitio

Aún así, cuando se aplica a proyectos considerados como factibles para XP, el conjunto de prácticas tiene algunos aspectos que han sido considerados controversiales. Por ejemplo:

- XP desalienta el diseño, sobre todo el de una arquitectura inicial completa
- XP es débil en la documentación
- El modelo XP no aplica para proyectos donde la seguridad es crítica
- El exceso de pruebas retrasa el desarrollo
- El diseño simple solo aplica a proyectos simples
- La programación en pares consume mayor tiempo y recursos
- La propiedad colectiva del software es causa de problemas

También se dice que XP asume implícitamente que siempre se utiliza el enfoque de programación orientada a objetos, lo cual no es necesariamente válido y que en general faltan datos estadísticos sólidos que comprueben que la práctica de XP mejora el desempeño ofrecido por otros modelos, ya

que, según ciertos críticos, la evidencia actual recae mas en anécdotas que en abundancia de datos concretos.

Algunas prácticas, como la refactorización y la planeación, que se reconocen ampliamente como un valor positivo, son criticadas en XP por sobre-utilizarse. La refactorización, por ejemplo, se ve como sinónimo de rediseño constante y que se puede tomar como una excusa para relegar hasta el último minuto el diseño e irlo construyendo o modificando junto con el código. También la planeación, según algunos críticos, no debería hacerse sobre la marcha como parece recomendar XP, dada la experiencia de desastres en proyectos. Finalmente, y por ser la programación en pares quizás la práctica más polémica, merece un análisis más detallado. Se argumenta, por ejemplo, que no cualquier clase de programador desea trabajar de esta manera, dado que muchos prefieren trabajar solos cuando están operando en forma altamente creativa, y unicamente integrarse en equipo cuando hay piezas de información que compartir o decisiones que hacer.

En XP, se exponen algunas barreras para poder implementar ésta práctica, como son:

- Falta de comunicación
- Evaluaciones por desempeño individual en las organizaciones
- Idiosincrasia del programador típico, antisocial ó retraído
- Inadaptación al entorno físico sugerido como áreas abiertas en lugar de cubículos

3.2 Metodología AM

AM es una metodología basada en prácticas para una efectiva modelación y documentación de sistemas de software [?]. AM simplemente es una colección de valores, principios y prácticas para la modelación de software que puede ser aplicado en un proyecto de desarrollo de software en una efectiva y ligera manera. El secreto de la metodología AM no son las técnicas de modelado en sí mismas, tal como modelos de caso de uso, modelos de clases, modelos de datos, o modelos de interfaces de usuario, pero si cómo son aplicadas. Se deberá tener un modelado ágil enfocándose en los requerimientos, análisis, arquitectura y diseño.

AM no define detalladamente los procedimientos para crear un determinado tipo de modelo, en lugar de ello provee consejos de cómo ser realmente efectivo con un modelo. AM no se trata de realizar menos modelos, de hecho frecuentemente se elaboran más modelos siguiendo la metodología AM que comparado con otras metodologías.

Los modelos ágiles son más efectivos que los modelos tradicionales ya que son justamente apenas lo suficientemente buenos, no tienen porque ser perfectos. Un prototipo en papel puede ser un modelo ágil, un bosquejo en un pizarrón acrílico también puede ser un modelo ágil, un diagrama en Visio igualmente. El tipo del modelo, o la herramienta con que fue creado realmente no importa, lo importante es que el modelo es justo lo suficientemente bueno para satisfacer la tarea específica.

Un concepto importante de entender acerca de AM es que no es un proceso completo de desarrollo de software, AM se enfoca en una efectiva modelación y documentación. No incluye actividades de programación, aunque puede considerarse la ejecución de pruebas como un modelo. AM no cubre la gestión de proyectos, actividades de despliegue, operaciones del sistema, soporte del sistema, y demás actividades relacionadas con esos temas. Ya que AM se enfoca en una porción de todo el proceso de desarrollo de software, se necesita usar con otros procesos completos tales como XP, DSDM⁵,

⁵Dynamic System Development Method — Método de Desarrollo de Sistemas Dinámicos

SCRUM⁶, o RUP. La Figura 3.2 muestra cómo AM puede ser utilizada para complementar otros procesos o metodologías de desarrollo.

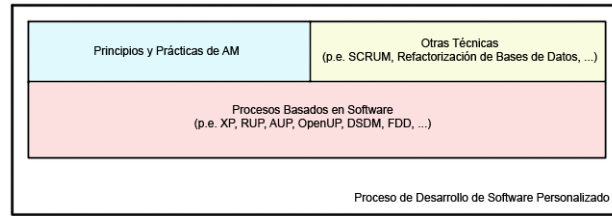


Figura 3.2: Agile Modeling. Tomado de <http://www.agilemodeling.com>

3.2.1 Principios

AM define una serie de principios básicos y complementarios que pueden ser aplicados en un proyecto de desarrollo de software para crear las bases para un conjunto de prácticas de modelado. Alguno de estos principios han sido adoptados por la metodología XP, que a su vez las adopta de técnicas comunes de ingeniería de software. La mayoría de los principios son presentados haciendo énfasis en sus implicaciones para el modelado.

Principios Básicos

Modelando con un propósito: muchos desarrolladores se preocupan sobre si sus artefactos, tales como modelos, códigos fuentes o documentos, son lo suficientemente detallados o si hay muchos detalles, o similarmente si ellos son lo suficientemente acertados. En lo que respecta a la modelación, quizás sea necesario entender mejor un aspecto del software a desarrollar, quizás sea necesario comunicar las propuestas a los altos cargos para justificar el proyecto, quizás se necesite crear documentación que describa el sistema desarrollado para las personas que lo usarán o mantendrán en el tiempo. El primer paso sería identificar un propósito válido para crear el modelo y la audiencia a quien está dirigido el modelo, luego basado en el propósito y la audiencia desarrollarlo desde el punto para que sea suficientemente acertado y detallado. Una vez cumplido con las metas del modelo estará completado por el momento y se deberá hacer otras cosas, como escribir algún código para mostrar cómo funciona el modelo. Este principio también se aplica para realizar cambios en un modelo existente, si se está realizando un cambio, quizás aplicando un patrón conocido, deberá existir un razón válida para realizar ese cambio (soportar un nuevo requerimiento o refactorizar el trabajo para hacerlo más limpio).

Maximizar el retorno de la inversión: las partes interesadas en el proyecto (clientes, desarrolladores, dueños del desarrollo) invierten recursos, tales como tiempo, dinero, facilidades, entre otros para tener un desarrollo de software que cumpla con todas las necesidades. Los interesados en el proyecto desean invertir sus recursos de la mejor manera posible y no malgastados por el equipo. Además, merecen tener la última palabra en la forma en que esos recursos se invierten o no.

Viajar ligero: cada artefacto que sea creado deberá ser mantenido en el tiempo. Si se decide, por ejemplo mantener 7 modelos y luego cuando cualquier cambio ocurra (nuevo requerimiento,

⁶Metodología para la gestión de proyectos basada en pequeñas iteraciones llamadas SPRINTS

actualización de un requerimiento, una nueva propuesta es tomada por el equipo de desarrollo, una nueva tecnología es adoptada, entre otros) se deberá considerar el impacto sobre los 7 modelos y luego actualizarlos acordemente. Ahora, si se decide mantener solo 3 modelos es claro que se tendrá menos trabajo para soportar el mismo cambio, siendo de esta manera más ágil en el desarrollo y mantenimiento. Igualmente cuanto más complejos y detallados sean los modelos, más probable es que cualquier cambio sea difícil de lograr. Un equipo de desarrolladores que decide desarrollar y mantener un detallado documento de requerimientos, una detallada colección de modelos de análisis, una detallada colección de modelos de arquitectura y una detallada colección de modelos de diseño rápidamente descubrirán que están gastando la mayor parte del tiempo en actualizaciones de documentos en lugar de escribir código.

Múltiples modelos: eventualmente se necesitará el uso de múltiples modelos de desarrollo de software ya que cada modelo describe un simple aspecto del software. Considerando la complejidad del software moderno, se necesitará un amplio rango de técnicas en el conjunto de herramientas de modelado para ser realmente efectivos. Un punto importante es que no todo el tiempo se necesitarán todos esos modelos para cualquier desarrollo de sistemas, pero eso depende de la naturaleza exacta del software que se esté desarrollando, será necesario tener por lo menos un subconjunto de los modelos. Con frecuencia se usarán más algunos tipos de modelos que otros.

Rápida retroalimentación: el tiempo entre una acción y la retroalimentación de esa acción es crítico. Si se trabaja con otras personas sobre un modelo, particularmente cuando se trabaja con una tecnología de modelado compartida, se obtiene casi instantáneamente una retroalimentación de las ideas. Trabajando cerca del cliente, para entender los requerimientos, analizar esos requerimientos o para desarrollar una interfaz de usuario, provee oportunidades para una rápida retroalimentación.

Asumir simplicidad: lo ideal es siempre hacer la solución lo más simple que se pueda. No sobrecargar el software, o en el caso de AM no representar características adicionales en el modelo que no se necesitan en el momento y actualizar el sistema en un futuro cuando los requerimientos evolucionen. Mantener el modelo tan simple como sea posible.

Adoptar cambios: los requerimientos evolucionan sobre el tiempo. Las partes interesadas en el proyecto (clientes, desarrolladores, dueños del desarrollo) pueden cambiar a medida que el proyecto avanza, nuevos desarrolladores son agregados y otros simplemente pueden salir. Los puntos de vistas pueden cambiar, potencialmente cambian las metas y los criterios de éxito del esfuerzo realizado. La consecuencia es que el ambiente de desarrollo cambia y como resultado las propuestas y aportes deberán reflejar esa realidad.

Cambio incremental: un importante concepto de entender con respecto a la modelación es que no tiene por qué hacerse bien la primera vez, de hecho, es muy poco probable que pueda hacerse incluso luego de haberlo intentado. Además no es necesario captar cada detalle en el modelo, sólo se necesita que sea lo suficientemente bueno en el momento. En lugar de tratar de desarrollar un modelo que abarque todo al comienzo, se deberá comenzar por el desarrollo de un modelo pequeño, o tal vez un modelo de alto nivel, y evolucionar a lo largo del tiempo (o simplemente descartarlo cuando ya no sea necesario) de manera incremental.

Calidad de trabajo: a nadie le gusta el trabajo descuidado. Las personas que realizan el trabajo no les gusta porque en ocasiones no se sienten orgullosos de él, las personas que luego realizan las actualizaciones y mantenimientos no les gusta porque es más difícil de entender y actualizar, y finalmente a los usuarios no les gusta el trabajo porque no cumple con sus expectativas.

Software de trabajo es el principal objetivo: la meta del desarrollo de software es producir un software de trabajo de alta calidad que satisfaga las necesidades de los interesados en el proyecto de una manera eficaz. La principal meta no es producir una documentación extraña, o extraños artefactos de gestión, o incluso modelos. Cualquier actividad que no contribuye directamente al logro de este objetivo debe ser evitado y cuestionado si no puede ser justificado.

Considerar que el próximo avance es el siguiente objetivo: el proyecto puede ser considerado un fracaso, aun cuando el equipo de desarrollo ofrezca un software de trabajo a los usuarios. Parte de satisfacer las necesidades de los interesados del proyecto es asegurar que el sistema sea lo suficientemente robusto como para que pueda ser extendido en el tiempo. En resumen cuando se está desarrollando un sistema es necesario mantener la atención en el futuro.

3.2.2 Prácticas

AM define un conjunto de prácticas básicas y complementarias [?], basadas en los principios antes descritos. Algunas de las prácticas han sido adoptadas por XP. La mayoría de las prácticas son presentadas haciendo énfasis en sus implicaciones para el modelado.

Prácticas Básicas

Participación activa de los participantes: AM amplía las prácticas de los interesados en el proyecto, incluyendo usuarios directos, administrador, los altos directivos, personal de operaciones, personal de soporte, los cuales participan activamente en el proyecto. Esto incluye la toma oportuna de decisiones de asignación de recursos por la alta gestión, el apoyo público y privado para el proyecto de los altos directivos, la participación activa del personal de operaciones y soporte en el desarrollo de las necesidades y modelos existentes en sus respectivos ámbitos.

Modelado con los demás: cuando se modela con un propósito a menudo se encuentra que se está modelando para entender algo, para comunicar ideas a los demás, o que se está tratando de desarrollar una visión común sobre el proyecto. Esta es una actividad de grupo, en la que es deseable el aporte de varias personas que trabajan juntos de manera efectiva. A menudo se encontrará que el equipo de desarrollo tiene que trabajar en conjunto para crear el grupo básico de modelos críticos del proyecto. Por ejemplo, para desarrollar la arquitectura del sistema, es necesario a menudo una modelación en grupo para desarrollar una solución en la cual todos están de acuerdo, así como una que sea lo más simple posible. La mayoría de las veces la mejor manera de hacerlo es hablar del tema con una o más personas.

Desarrollar los artefactos correctos: cada artefacto tiene sus propias aplicaciones específicas. Por ejemplo, un diagrama UML de actividad es útil para describir el proceso del negocio, mientras que la estructura de la base de datos está mejor representada con los datos físicos o con un modelo de persistencia de datos. Muy a menudo, un diagrama es una mejor elección que el código porque con frecuencia se pueden explorar alternativas de diseño de manera más efectiva por el dibujo de un par de diagramas de lo que se puede sentándose y desarrollando códigos de ejemplo.

Iterar a otro artefacto: cuando se está trabajando en un artefacto de desarrollo, como por ejemplo un caso de uso, diagrama de secuencia, o incluso el código fuente, y se encuentra algún tipo de inconveniente, entonces se debería considerar la posibilidad de trabajar en otro artefacto por el momento. Cada artefacto tiene sus fortalezas y debilidades, cada artefacto es bueno para un determinado tipo de trabajo. Por ejemplo, si se está trabajando en un caso de uso esencial,

quizá se quiera considerar la posibilidad de modificar el enfoque para empezar a trabajar en un prototipo de la interfaz de usuario, una regla de negocio, un sistema de caso de uso, entre otros. Al pasar a otro artefacto casi inmediatamente se comenzarán a resolver los conflictos porque se está haciendo progreso de trabajo en ese otro artefacto.

Usar las herramientas más simples: la gran mayoría de los modelos se pueden extraer en pizarrón o en papel. Cada vez que se desea guardar uno de estos diagramas se puede tomar una foto de él con una cámara digital, o incluso simplemente transcribir al papel. Esto funciona porque la mayoría de los diagramas son desaprovechados; su verdadero valor proviene del dibujo pensado para un tema, y una vez que el problema se resuelva el diagrama no ofrece mucho valor. Es recomendado utilizar una herramienta de dibujo para crear diagramas y presentarlos a las partes interesadas en el proyecto y, en ocasiones, utilizar una herramienta de modelado ya que proporcionan valor a las actividades de programación, tales como la generación de código.

Modelación en pequeños incrementos: aplicar un desarrollo incremental en el que se organiza un gran esfuerzo en porciones más pequeñas que serán liberadas y actualizadas en el tiempo, lo cual conlleva a que luego de varias semanas, un mes o dos, aumente la agilidad, permitiendo entregar el software en manos de los usuarios con mayor rapidez.

Una sola fuente de información: la información debe almacenarse en un lugar único. En otras palabras, no sólo se debe desarrollar los artefactos correctos. También se debería almacenar la información en el mejor lugar posible. A veces el mejor lugar para almacenar la información es en un documento ágil, que a menudo es el mismo código fuente.

Propiedad colectiva: todos pueden trabajar en cualquier modelo, y de hecho en cualquier artefacto del proyecto, en caso de que lo necesiten.

Crear varios modelos en paralelo: debido a que cada tipo de modelo tiene sus fortalezas y debilidades no hay un modelo único que sea suficiente para las necesidades de modelado. Por ejemplo, cuando se están explorando los requerimientos puede que se necesite desarrollar algunos casos de uso o historias de usuario, un prototipo de la interfaz de usuario, y algunas reglas de negocio. En combinación con la práctica de iterar a otro artefacto, a menudo los grupos de desarrolladores descubren que se es mucho más productivo trabajando en varios modelos al mismo tiempo que si sólo se centran en uno.

Crear contenido simple: es necesario mantener el contenido real de los modelos, necesidades, análisis, arquitectura, diseños lo más simple que se pueda sin dejar de satisfacer las necesidades de los interesados en el proyecto. La consecuencia es que no se debe añadir más aspectos a los modelos a menos que sean justificables. Por ejemplo, si no existe la obligación de añadir funciones de auditoría al sistema entonces no añadir esas funciones a los modelos.

Mostrar públicamente los modelos: una buena práctica es la de mostrar los modelos públicamente. Esto apoya la comunicación abierta y honesta entre el equipo de trabajo porque todos los modelos actuales son rápidamente accesibles a ellos, así como con los altos cargos del proyecto, ya que no se está escondiendo nada a ellos.

3.2.3 Aspectos Positivos

Algunos de los beneficios de AM según [?], son:

Rentabilidad: AM define un conjunto de valores, principios y prácticas relativos a la eficacia ligera de modelado y documentación. Mediante la creación de modelos y documentos que son apenas lo suficientemente bueno para maximizar el retorno de la inversión de interesados.

Se definen explícitamente las técnicas de los proyectos ágiles: AM se refiere al tema de cómo los desarrolladores de modelos y documento ágiles sobre los proyectos de software adoptan un enfoque ágil como XP, DSDM, o SCRUM. Citando a [?]: “Vamos a seguir perfeccionando el diseño de un sistema, a partir de un principio muy simple. Vamos a eliminar cualquier flexibilidad que no resulte útil”. Contrariamente a las afirmaciones de algunos de los detractores de XP es posible, de hecho, invertir tiempo de modelado cuando usamos un enfoque XP, pero sólo cuando no se tiene otra opción. A veces es mucho más productivo para un programador extraer algunas burbujas y líneas de pensamiento para una idea, o para comparar diferentes enfoques para resolver un problema.

Mejora el modelado y la documentación sobre otros procesos: AM se refiere al tema de cómo hacer la modelación de manera eficaz en un proyecto. AM ayuda a mejorar la eficacia de los esfuerzos de modelado sobre un proyecto.

3.3 Resumen

XP es un proceso de desarrollo de software que surge como un proyecto liderado por Kent Beck y, que se basa en cuatro fundamentos básicos: comunicación, simplicidad, feedback y coraje. El secreto del éxito de XP radica en su desarrollo orientado en requerimientos cambiantes o dinámicos. Está claro que los cambios en los requerimientos a lo largo del desarrollo siempre estarán presentes y XP contempla esa realidad y saca el mayor provecho al minimizar el impacto de un cambio en los requerimientos.

Para lograr sus objetivos XP contempla una serie de principios, prácticas y valores básicos. Al aplicar XP en un proyecto de desarrollo, éste debe ser más flexible con respecto al cambio.

Por su parte, AM es una metodología basada en prácticas para una efectiva modelación y documentación de sistemas de software. AM simplemente es una colección de valores, principios y prácticas para la modelación de software que puede ser aplicado en un proyecto de desarrollo de software en una manera efectiva y ligera. XP y AM están relacionados a través de algunos principios, prácticas y valores básicos comunes, lo que permite que puedan ser combinados y así producir software de mayor calidad.

Capítulo 4

Sistemas RFID

RFID es una tecnología muy versátil y prometedora que puede ser aplicado en varios escenarios, obteniendo excelentes resultados. Esto hace que esta tecnología se coloque a la vanguardia en la mayoría de los sectores de la vida cotidiana: negocios, seguridad, turismo, gobierno, alimentos, etc.

Actualmente esta tecnología sigue en desarrollo y cada vez más se le pueden conseguir nuevas aplicaciones. Sin embargo, para seguir investigando sobre las potencialidades de ésta tecnología es necesario comprender cómo está compuesto un sistema básico RFID. En este capítulo se intentan explicar los aspectos más relevantes de los principales componentes de un sistema RFID. Además, se explican los aspectos relacionados con las aplicaciones y la seguridad de los sistemas RFID. Finalmente, este capítulo expone la visión de la actualidad y el futuro de los sistemas RFID.

4.1 Componentes

Como se describió en el capítulo anterior, se pueden distinguir tres componentes esenciales en un sistema RFID: las etiquetas, los lectores y el middleware. En el caso de una tienda o comercio minorista que use un sistema RFID, las etiquetas RFID representan la mercancía etiquetada. Los lectores RFID se pueden encontrar en las salidas y los puestos de chequeo. Estos lectores deben tener la capacidad de leer miles de etiquetas por minuto. Los lectores deben ser configurados y administrados de tal manera que cubran los lugares de la tienda en los que los lectores pueden fallar. El RFID middleware representa uno o varios módulos de software que se encargan de administrar las responsabilidades antes descritas.

Otros componentes como las aplicaciones de borde representan aplicaciones empresariales que contienen componentes ejecutándose con fines específicos para la tienda. Otros componentes que se pueden destacar en sistemas RFID son los servicios de información RFID que se encargan de almacenar eventos y otros datos importantes.

La infraestructura de estos sistemas define una red de información compuesta por subsistemas de borde, centros de datos y aliados comerciales, entre otros. En la Figura 4.1 se pueden distinguir otros dos componentes:

El bus de servicios empresariales: representa cualquier mecanismo seleccionado por la empresa para la integración de las aplicaciones.

Las aplicaciones empresariales: representan las aplicaciones que son clientes o son afectadas por datos RFID en la empresa.

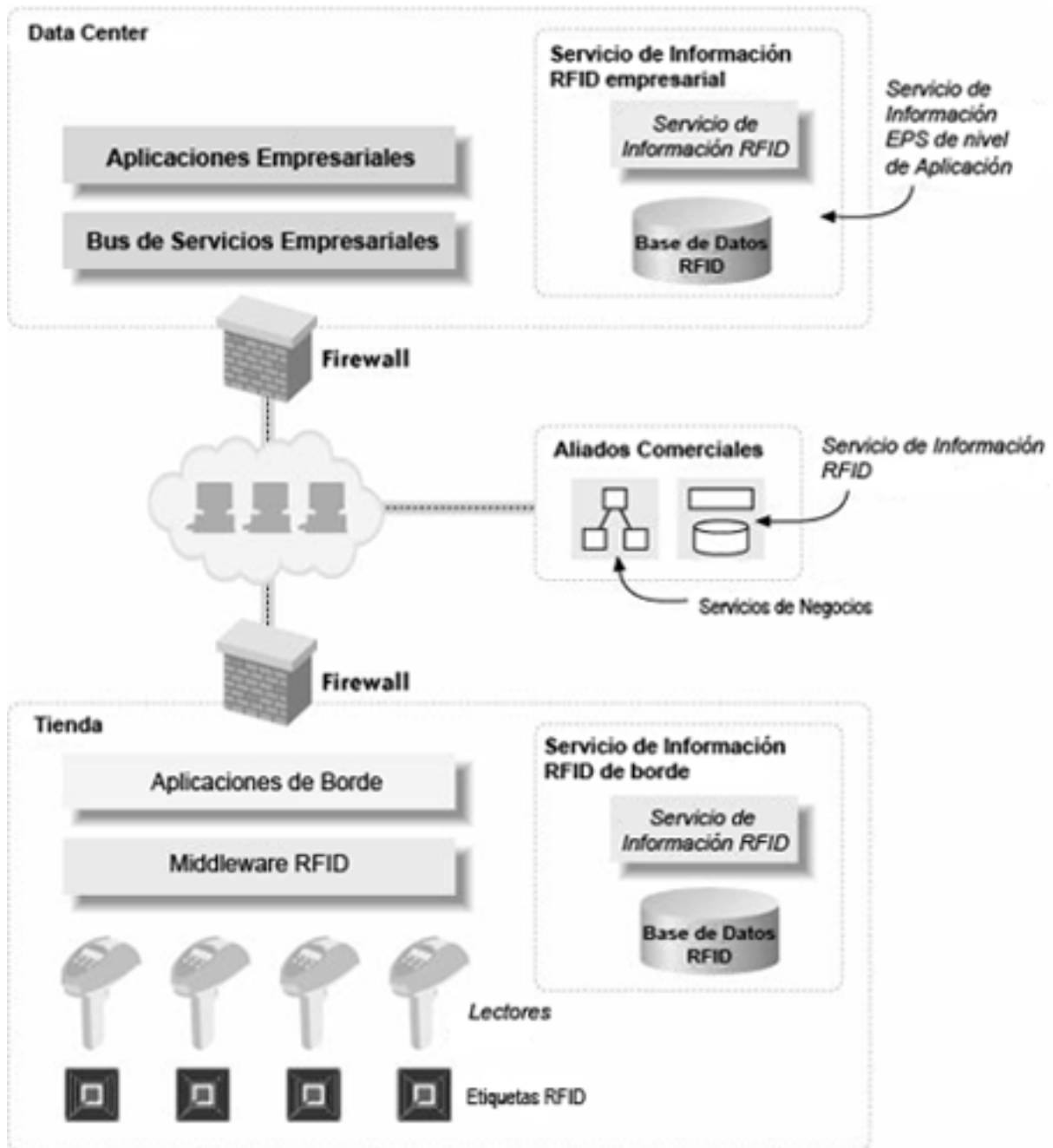


Figura 4.1: Componentes de un Sistema RFID. Tomado de [?]

A continuación, se describen algunos de los aspectos más importantes de los componentes de los sistemas RFID [?]:

4.1.1 Etiquetas

Las etiquetas permiten que los lectores puedan identificar objetos al transmitir señales de radio a una frecuencia predefinida y a intervalos (usualmente, cien veces por segundo). Cualquier etiqueta que esté

en la frecuencia y alcance del lector, recibe sus transmisiones. Las etiquetas usan la energía de la señal del lector para reflejar de vuelta esta señal al lector y además pueden modular esta señal para enviar información (por ejemplo, un número único de identificación). En la Figura 4.2 se puede apreciar este escenario.

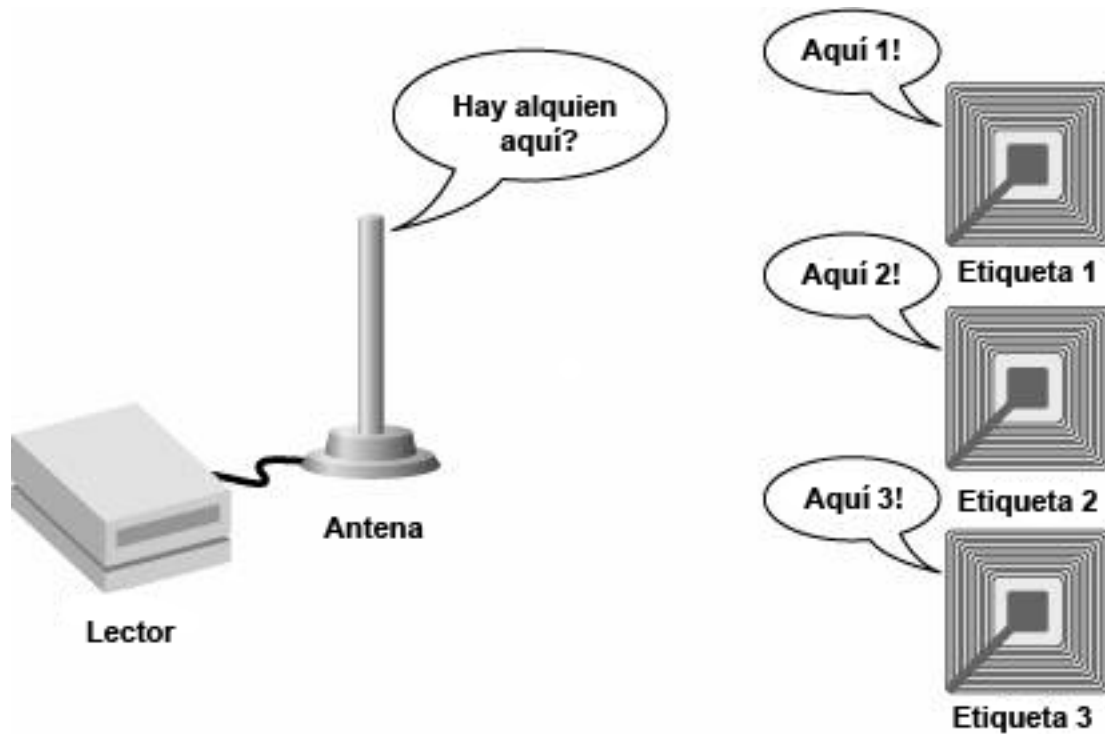


Figura 4.2: Comunicación entre Etiquetas y Lectores RFID. Tomado de [?]

Existen diferentes tipos de etiquetas y lectores que se adaptan a diferentes tipos de aplicaciones y ambientes. Debido a estos es posible conseguir una gran variedad de etiquetas y lectores que pueden variar en cuanto al costo y características. A continuación se describen algunas de las características más importantes de las etiquetas RFID:

Presentación: las etiquetas RFID se pueden encontrar en formas de botones PVC, viales de vidrio, etiquetas de papel o etiquetas plásticas. También se pueden encontrar en joyas, llaveros, incorporados en una llave que puede servir como mecanismo de inmovilización de vehículos.

Uniones: se refiere a los medios por los que la etiqueta y el lector se comunican. Diferentes métodos afectan aspectos como el rango de comunicación, el precio y las condiciones que pueden causar interferencias.

Energía: muchas etiquetas RFID usan un sistema pasivo que permite que la etiqueta reciba energía obteniéndola de un campo electromagnético o un pulso de energía de radiofrecuencia emitido por el lector. Otros tipos de etiquetas usan un sistema activo, el cual usa una batería para suministrar energía al chip y los sensores de la etiqueta. Sin embargo, estas etiquetas también utilizan la energía del lector para la comunicación. El último tipo de etiqueta usa su propia energía para las comunicaciones e incluso es capaz de comunicarse con otras etiquetas sin necesidad de un lector.

Capacidad de almacenamiento de información: las etiquetas proveen una gran variedad de capacidades de almacenamiento. Existen etiquetas de sólo lectura que permiten escribir información una vez y otras que permiten escribir varias veces. Algunas otras son capaces de obtener información por su cuenta (como, por ejemplo, lecturas de temperatura o presión). Existen etiquetas que permiten almacenar 1 bit información usadas para la prevención de robos. Otras etiquetas permiten almacenar miles de bytes de información usadas en líneas de autoensamblaje.

Estándares: muchos sistemas RFID conforman estándares nacionales e internacionales particulares. Algunos estándares especifican frecuencias, tipos de uniones, capacidades de almacenamiento de información y más.

Es posible comprar algunos de estos dispositivos básicos y los precios varían dependiendo del constructor y los materiales empleados. En la Figura 4.3 se puede apreciar una pequeña variedad de dispositivos RFID que se pueden comprar por Internet a través del portal <http://www.trossenrobotics.com>.



Figura 4.3: Kit RFID

4.1.2 Lectores

Los lectores, también llamados interrogadores, son usados para reconocer la presencia de una etiqueta RFID cercana. El lector transmite energía en forma de radiofrecuencia a través de una o más antenas. La antena de una etiqueta cercana recibe esta energía y la convierte en energía eléctrica vía inducción electromagnética. Esta energía es suficiente para alimentar al chip semiconductor adherido a la antena de la etiqueta, la cual almacena la identidad de la etiqueta. Entonces la etiqueta envía de vuelta al lector su identidad, aumentando y disminuyendo la resistencia de la antena, como una especie de clave Morse. Esta es la manera típica de interacción entre etiquetas y lectores. Sin embargo, existen otros mecanismos que difieren levemente a este esquema.

Los lectores se pueden encontrar en diferentes formas y tamaños. También pueden encontrarse lectores de tipo estacionarios, así como también de tipo portable. La Figura 4.4 muestra los componentes de un lector y cómo se ubican los lectores entre el mundo externo y las etiquetas.

A continuación se describen los componentes de los lectores RFID:

API del lector: el API permite a los programas registrar y capturar eventos de lectura de etiquetas RFID. También provee la posibilidad de configurar, monitorear y administrar el lector.

Comunicaciones: los lectores son dispositivos de borde que al igual que cualquier otro dispositivo RFID están conectados a la red del sistema de borde. El componente de comunicaciones maneja las funciones de red. La Figura 4.5 muestra el borde de un sistema RFID:



Figura 4.4: Partes de un Lector RFID. Tomado de [?]

Manejador de eventos: cuando un lector detecta una etiqueta, se denomina una observación. Una observación que difiere de observaciones previas se denomina un evento. El análisis de observaciones se denomina filtrado de eventos. El Manejador de Eventos define qué tipo de observaciones son consideradas eventos y determina qué eventos son lo suficientemente importantes para ser reportados o ser enviados inmediatamente a aplicaciones externas en la red.

Subsistema de antenas: el subsistema de antenas consiste de una o más antenas y las interfaces de soporte y lógica que permiten a los lectores interrogar etiquetas RFID.

4.1.3 Middleware

Escoger las etiquetas y lectores adecuados y determinar dónde ubicar las antenas es sólo el primer paso para la construcción de un sistema RFID. La posibilidad de leer millones de etiquetas que pasan por la cadena de suministros y la necesidad de asociar códigos a información importante, genera grandes cantidades de datos con relaciones complejas. Uno de los principales beneficios del uso del middleware es que estandariza los mecanismos para manejar las grandes cantidades de información que producen estas etiquetas. Adicionalmente al filtrado de eventos, también es necesario un mecanismo para encapsular las aplicaciones para prevenir que conozcan los detalles de la infraestructura (lectores, sensores y su configuración). Lo ideal es tener interfaces de nivel de aplicación basadas en estándares que puedan ser usadas por las aplicaciones para obtener observaciones RFID importantes. La Figura 4.6 muestra los componentes principales del middleware RFID:

Adaptador de lectura: existen muchos tipos de lectores RFID en el mercado actual y cada uno tiene su propia interfaz. Sería impráctico que los desarrolladores de aplicaciones tengan que aprender diferentes tipos de interfaces. Las interfaces de lectores, el acceso a datos y la administración de dispositivos difieren mucho. Por lo tanto, se debe tratar de usar un middleware que evite tener que aprender las idiosincrasias de cada lector. El adaptador de lectura provee una capa de

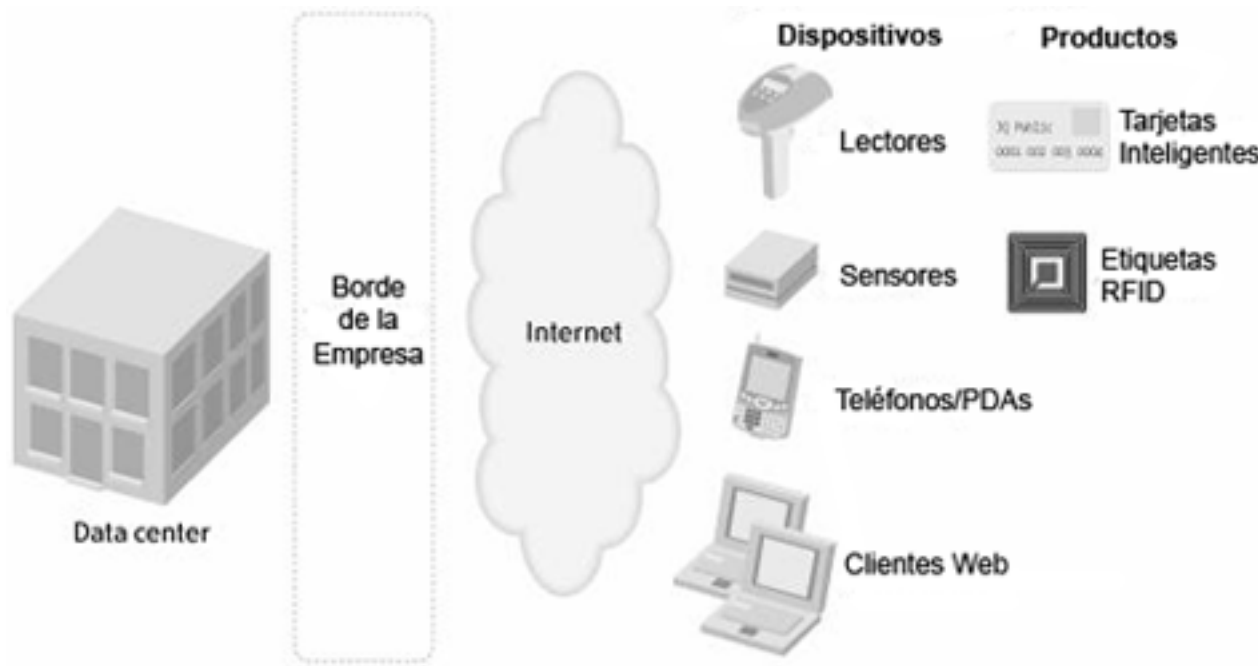


Figura 4.5: El Borde Empresarial en un Sistema RFID. Tomado de [?]



Figura 4.6: Componentes del Middleware RFID. Tomado de [?]

abstracción que encapsula las interfaces propietarias de cada lector de manera que no entren en contacto con los desarrolladores de aplicaciones.

Manejador de eventos: una cadena de suministros completamente funcional y con un sistema RFID habilitado puede registrar miles de eventos de lectores interrogando a razón de cientos de lecturas por minuto. Muchas de estas observaciones pueden ser de muy fina granularidad para ser importantes para las aplicaciones. Por lo tanto se quiere encapsular la interfaz de lectura para prevenir que las aplicaciones sean abrumadas por datos no procesados. Por lo tanto, las empresas necesitan desplegar un middleware RFID de propósito especial cerca de los bordes de la infraestructura de información tecnológica.

Los lectores tienen una precisión menor del 100% al leer etiquetas en sus alrededores. Suponiendo

un escenario en el que hay 100 productos cerca de un lector configurado para interrogar 200 veces por minuto, las probabilidades son que cada interrogación capture entre 80 y 99 de esos productos. Este escenario demuestra por qué los escaneos de lectores RFID son considerados crudos y requieren un mayor procesamiento para conseguir eventos de negocio importantes.

Consideremos otro escenario en el que se tiene una cadena de tiendas que implementa un sistema de anaqueles inteligentes. Suponiendo que la tienda calcula 25 productos por anaquel y 4 anaqueles por pasillo, para un total de 100 productos por pasillo. Suponiendo además que la cadena cuenta con 10 tiendas y cada una con 20 secciones que contienen 20 pasillos. Así pues, cada tienda tiene 400 anaqueles, lo que significa que cada tienda maneja un inventario promedio de 40000 productos.

Ahora bien, si se considera el volumen de información generado a medida que el sistema RFID lee estos datos, se pueden obtener los siguientes resultados:

- Cada interrogación retorna observaciones que contienen información de todos los productos reconocidos por estar en los anaqueles seleccionados
- 25 productos por anaquel, multiplicado por 4 anaqueles por pasillo, multiplicado por 10 interrogaciones por minuto, es igual a 1000 observaciones por minuto, por pasillo
- 1000 productos leídos por minuto, multiplicado por 400 pasillos, es igual a 400000 observaciones por minuto
- 400000 productos leídos por minuto, multiplicado por 60 minutos por hora, es igual a 2400000 observaciones por hora
- Suponiendo que la tienda abre 10 horas al día. 10 horas, multiplicado por 2400000 productos leídos por hora, es igual a 24000000 de observaciones por día, por tienda
- En 10 tiendas, 240000000 de observaciones diarias en todas las tiendas

Este escenario demuestra la necesidad de mecanismos de observación en conjunto, por tiempo y a través de múltiples lectores. También demuestra la necesidad de filtrar, consolidar, y transformar las observaciones no procesadas de los lectores. Es por esto que los sistemas RFID requieren un middleware que se ejecute dentro de los bordes de los centros de datos de la empresa. De esta manera, sólo las observaciones que son importantes para las aplicaciones corporativas son enviadas a esas aplicaciones y el resto de los datos es filtrado por el middleware.

Ahora bien, considerando el caso en que se coloquen dos antenas por pasillo, sus rangos de lectura pueden solaparse. Por lo tanto, las observaciones provenientes de estas antenas deben ser filtradas para eliminar lecturas duplicadas. También, como las interrogaciones individuales tienden a ser menos de 100% precisas, estas observaciones necesitan ser agrupadas a lo largo de varios ciclos de lectura.

Interfaz de nivel de aplicación: la interfaz de nivel de aplicación es la última capa en el middleware RFID. Su principal propósito es proveer un mecanismo estandarizado que permita a las aplicaciones registrar y recibir eventos filtrados de un grupo de lectores. Adicionalmente, la interfaz de nivel de aplicación provee un API estándar para configurar, monitorear y administrar el middleware RFID y los lectores y sensores que éste controla. Muchos productores de middlewares RFID proveen interfaces propietarias diseñadas para estos propósitos.

4.1.4 Bus de Servicios RFID

Un ESB¹ es una plataforma distribuida de integración diseñada para la conectividad entre aplicaciones, transformación de datos, transacciones garantizadas y mensajería. Un bus de servicios RFID es un tipo de ESB usado para integrar aplicaciones que manejen datos RFID. Dependiendo de la implementación, un producto ESB puede proveer servicios web, mensajería, orquestación de procesos de negocio, entre otros servicios. Generalmente los ESB orquestan procesos de negocios a través de aplicaciones.

Los sistemas RFID deben coexistir con y extender las funcionalidades de aplicaciones empresariales como sistemas WMS², sistemas ERP³ o sistemas de puntos de venta.

Un bus de servicios RFID es esencialmente un servidor de integración que ejecuta flujos de trabajo y provee mecanismos de integración usando módulos de aplicaciones empresariales. El bus de servicios RFID también se puede integrar a un ESB y puede ser configurado para proveer eventos y observaciones específicas.

Servicio de Información RFID

Es importante destacar que un EPC⁴ o cualquier otro sistema de identificación provee solamente un identificador único que por sí solo no provee mayor información acerca de un producto. Existen organizaciones que buscan, de manera colaborativa con las empresas y la industria tecnológica, establecer una red de servidores de Servicios de Información EPC (EPCIS) para proveer un repositorio de información sobre EPCs individuales. La información que estos servidores proveerían puede incluir la ubicación de la última observación de un producto, información sobre el precio, manuales o cualquier otra información apropiada. Pero esta visión puede tomar años en establecer una red de servidores EPCIS para toda la industria, por lo que se hace necesario un paso intermedio para implementar una base de datos que mapee los identificadores usados en los sistemas de seguimiento RFID a información más completa y relevante para las aplicaciones de negocio.

Red de Información RFID

La Red EPCglobal contempla un conjunto de estándares que buscan proveer un framework para el intercambio de información que se centra en la tecnología RFID y la infraestructura de Internet existente. Esta red ofrece mayor precisión y eficiencia en el seguimiento de productos entre aliados comerciales. En la Figura 4.7 se puede apreciar una visión de la red de información RFID:

Cuando un producto es manufacturado y enviado a una tienda, se registra la información de rastreo de ese producto. El EPCIS y el ONS⁵ controlan la manera en que el distribuidor captura esta información de rastreo y la relaciona con la información de rastreo provista por la manufacturera y luego con la tienda. De tal manera que un servidor EPCIS puede alojar y facilitar el acceso a información serializada de productos, lo que significa que el servidor EPCIS de un distribuidor puede estar configurado para establecer una interfaz con el servidor EPCIS de una manufacturera para acceder o agregar datos EPC relevantes.

El ONS es un directorio centralizado que asigna rutas a peticiones de información de productos EPC, siguiendo las mismas estrategias de un DNS⁶. La empresa VeriSign mantiene la raíz del servidor

¹Enterprise Service Bus - Bus de Servicios Empresariales

²Warehouse Management - Administración de Almacenes de Datos

³Enterprise Resource Planning - Planificación de Recursos Empresariales

⁴Electronic Product Code - Código Electrónico de Producto

⁵Object Naming Service - Servicio de Nombres de Objetos

⁶Domain Name System - Sistema de Nombres de Dominio

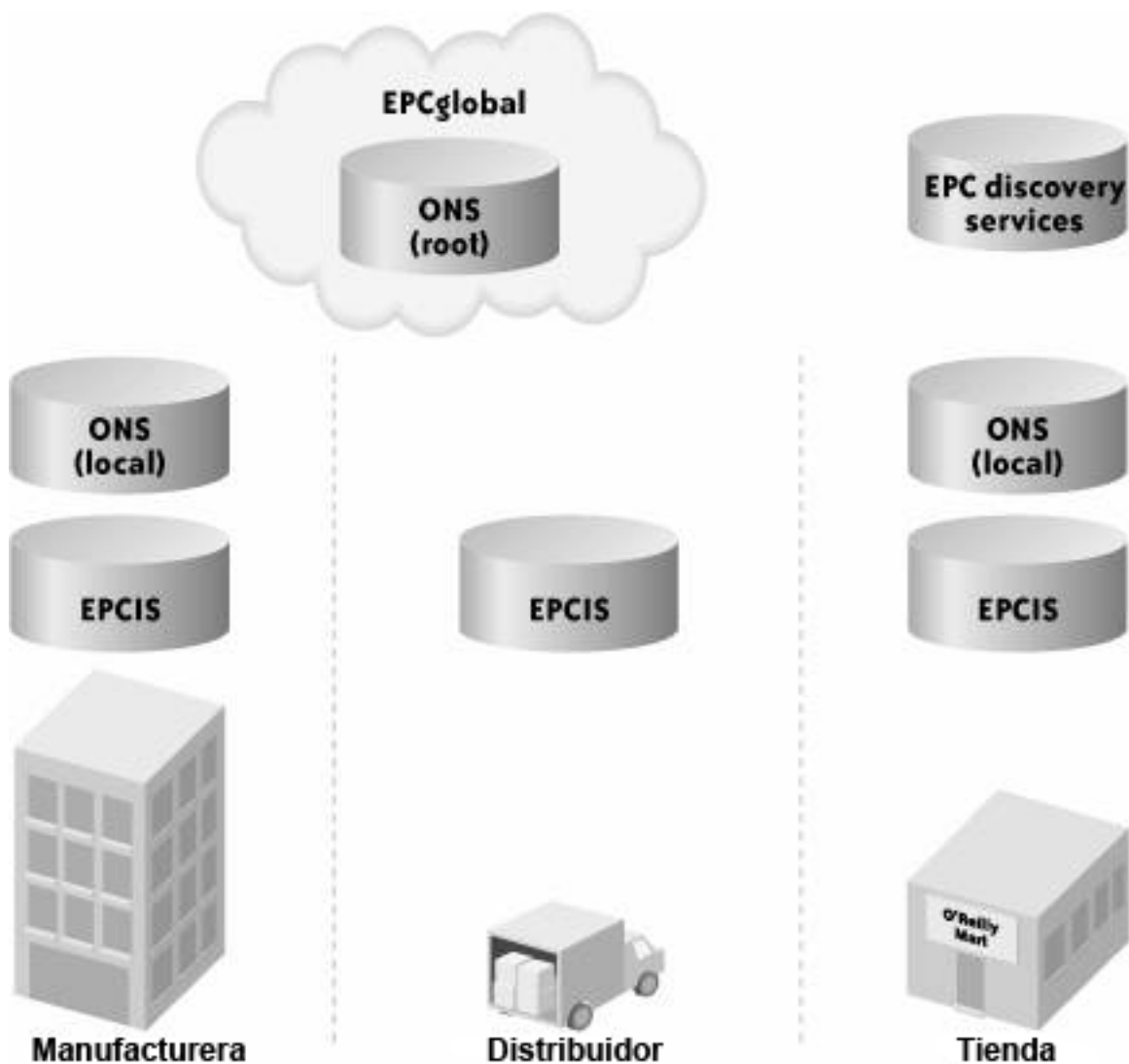


Figura 4.7: Red de Información RFID. Tomado de [?]

ONS para la red EPCglobal. El EPC Discovery Service es un registro de todos los servidores EPCIS que tienen información acerca de un producto en particular.

4.2 Aplicaciones

RFID es una tecnología versátil que permite ser usada tanto por el sector comercial como por gobiernos. El uso de RFID está creciendo constantemente de tal manera que el comercio y sus asociaciones, e incluso agencias gubernamentales anuncian semanalmente nuevos usos. A continuación se listan algunas de las aplicaciones más comunes que se pueden encontrar actualmente, principalmente en EEUU [?]:

Administración de materiales e inventarios: es una aplicación importante para la administración eficiente de la cadena de suministros, distribución y venta de muchos comercios. Las compañías

más conocidas por aplicar esta tecnología para fines son Wal-Mart, el Departamento de Defensa de EEUU, Tesco y Metro Gourp, entre otras.

Etiquetado de productos en anaqueles: esta aplicación es una manera práctica de proveer al consumidor información oportuna acerca de un producto. Marks & Spencer, Prada y Tesco son algunos de los comercios que han tenido éxito aplicando la tecnología RFID.

Tarjetas inteligentes: son utilizadas como una manera muy conveniente en el sector del transporte. El sistema SmartTrip del Metro de Washington es un caso de uso de esta tecnología.

Sistemas de pago: algunas tarjetas de crédito (por ejemplo, Mastercard PayPass, Chase Bank's Blink Mastercard y ExpressPay de American Express) pueden ser usadas en algunos puntos de venta (por ejemplo, el Mobil SpeedPass) para hacer los pagos más rápidos.

Logística: Kimberly Clark es una marca reconocida por usar la tecnología RFID para mejorar las actividades de logísticas de su cadena de valor.

Rastreo de bienes: es un mecanismo eficiente para el rastreo de mercancía en contenedores transportados vía marítima. Por ejemplo, las empresas Robert Bosch Tool, Coors UK y NASCAR Goodyear se sirven de esta tecnología.

Sistemas de encendido automático de vehículos: las compañías Toyota, Lexus y Audi, entre otras, ya están incorporando sistemas RFID para permitir a los usuarios un mecanismo conveniente de encendido de vehículos.

Deportes: muchas de los organizadores de eventos deportivos usan sistemas RFID para, por ejemplo, ubicar maratonistas u otros participantes durante eventos competitivos.

Boletos para eventos: la tecnología RFID se ha utilizado en eventos deportivos como el torneo Tennis Master Cup 2005, en la exposición Cannon 2005 en París y se tiene provisto en las Olimpiadas 2008 en Beijing.

Control de acceso: esta aplicación provee un mecanismo de seguridad y control de acceso en edificios, habitaciones, ciudades universitarias. Algunos de los proveedores de estos mecanismos son Texas Instruments e Idesco Oy.

Identificación de mascotas: es un mecanismo conveniente para identificar mascotas al implantar un microchip RFID contenido en una cápsula de vidrio en la piel del animal. Los mayores proveedores de estos productos son Avid y Home Again.

Etiquetado de ganado y animales salvajes: es una aplicación que asegura el rastreo e identificación de animales, para distinguir los animales sanos de los enfermos o que puedan estar contaminados. Los mayores proveedores de estos productos son Allflex, Digital Angel y Aleis International.

Etiquetado de personas: es un mecanismo que se usa principalmente con propósitos médicos y de seguridad (por ejemplo, para la prevención de secuestros de niños en hospitales). El principal proveedor de estos productos es VeriChip.

Rastreo de equipaje: el aeropuerto de Hong Kong, la aerolínea Delta Airlines y Globalbagtag son algunas de las instituciones que ya están utilizando la tecnología RFID para la recuperación y monitoreo de equipaje.

Pasaportes y control fronterizo: EEUU, Japón, Holanda, Noruega, Pakistán y Malasia son algunos de los países que utilizan sistemas RFID para el control de inmigrantes.

Bibliotecas: la Biblioteca del Vaticano, la Biblioteca de Berkeley y la Biblioteca de la Universidad de Connecticut han implementado sistemas RFID para proveer un mecanismo de seguridad y rastreo de libros.

Seguimiento de archivos: es una aplicación muy útil para el seguimiento de expedientes (por ejemplo, en oficinas asesoría legal)

Prevención de medicamentos falsos: Purdue Pharma's OxyContin es una empresa que usa un sistema RFID para prevenir la reproducción de medicamentos y los medicamentos genéricos de mala calidad.

Es posible distinguir una jerarquía entre los distintos tipos de aplicaciones RFID. La Figura 4.8 muestra la relación que existe entre las tecnologías y usos involucrados en las aplicaciones RFID:

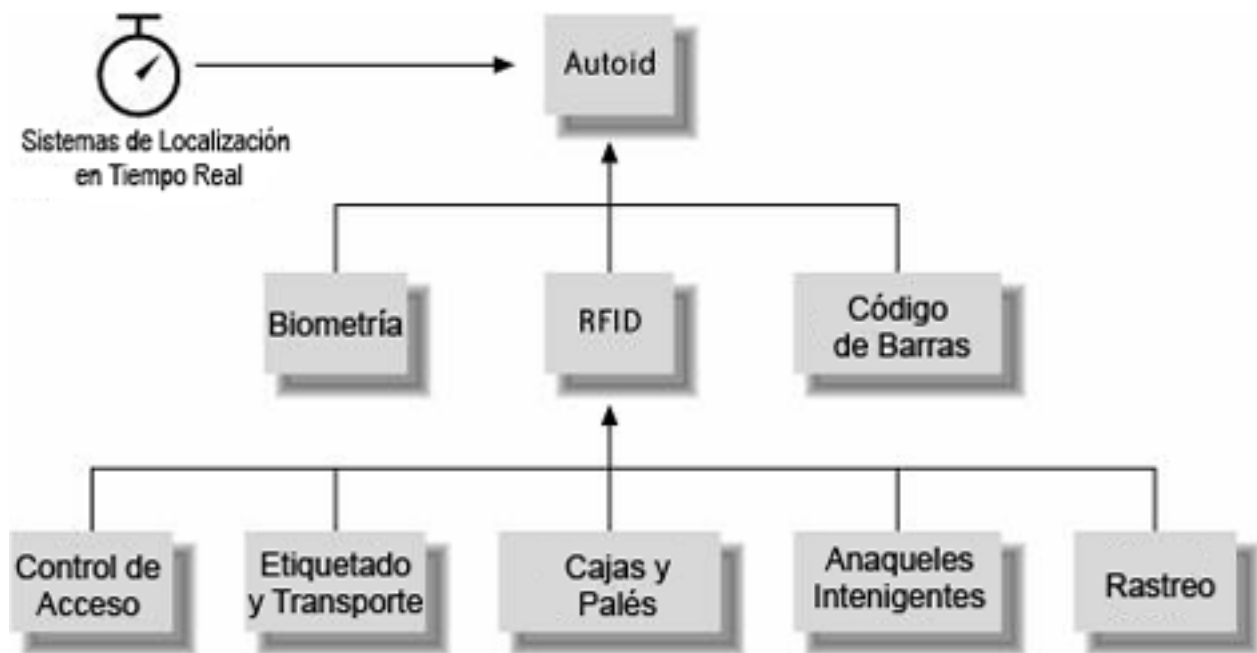


Figura 4.8: Relaciones entre los tipos de Aplicaciones RFID. Tomado de [?]

4.3 Seguridad y Privacidad

Hoy en día se utilizan en todo el mundo las etiquetas y chips de identificación por radiofrecuencia entre los que pueden citarse una serie de ejemplos donde la tecnología RFID se ha aplicado en las empresas y laboratorios como claves de acceso, para encender carros y como dispositivos de rastreo de inventarios. Ciertos fabricantes de medicinas se basan en las etiquetas RFID para rastrear los productos farmacéuticos. En un futuro cercano, las etiquetas RFID serán mucho más personales. La próxima generación de pasaportes y tarjetas de crédito en Estados Unidos contienen etiquetas RFID,

y la industria médica está estudiando el uso de etiquetas implantables para gestionar los pacientes de manera eficaz.

Las etiquetas trabajan enviando constantemente mensajes con bits de información a los lectores electrónicos especializados. La mayoría de las etiquetas RFID comerciales son emisoras pasivas, por lo que necesitan ser activadas por los lectores. Una vez activadas, las etiquetas emiten su señal indiscriminadamente dentro de un cierto rango, por lo general desde unos cuantos centímetros hasta unos pocos metros. Sin embargo, algunas etiquetas activas RFID que funcionan con baterías internas pueden enviar señales a cientos de metros. Ejemplos de estas son las que están desplegadas en peajes de pago automático.

Para protección de la información, las señales RFID pueden ser encriptadas mediante algoritmos adecuados. Las etiquetas que se utilizan para aplicaciones con manejo de información crítica, es probable que usen algún código cifrado para que sea difícil para los lectores no autorizados recuperar información (puede ser información como el nombre de una persona, edad, nacionalidad y foto y otras informaciones sensibles). Pero la mayoría de las etiquetas RFID comerciales no incluyen la seguridad ya que incrementa los costos.

Esto deja a la mayoría de las etiquetas RFID vulnerables a la clonación y la manipulación de datos, sobre todo si la etiqueta RFID tiene un área de memoria escribible. Las etiquetas RFID que se utilizan para rastrear envíos de productos o equipo caros, por ejemplo, suelen contener los precios e información del producto. Estas áreas de escritura pueden ser bloqueadas, pero a menudo son ignoradas, ya sea porque las empresas que utilizan RFID no conocen el funcionamiento de las etiquetas, o los campos de datos necesitan ser actualizados con alta frecuencia. De cualquier manera, estas etiquetas están abiertas a intrusiones de piratas informáticos o la manipulación de datos.

Vulnerar una etiqueta RFID es, en ocasiones, muy fácil. Se puede robar la tarjeta inteligente, plagiar el pasaporte de alguien e incluso clonar la etiqueta incrustada en el dispositivo. Existen muchos ejemplos de sistemas RFID que han sido vulnerados, se han registrados ataques de plagio de información personal que luego es usada con fines no éticos como ataques de forjamiento de datos, ataques de destrucción de información, entre otros. Estos ataques en la seguridad de etiquetas RFID y muchos más seguirán agregándose a la lista en los próximos años.

4.3.1 Problemas

Los problemas de seguridad con RFID según [?] pueden dividirse en las tres siguientes categorías:

- Problemas relacionados con la tecnología
- Problemas relacionados con la privacidad y la ética
- Problemas relacionados con la seguridad

4.3.2 Problemas Relacionados con la Tecnología

Estándares RFID

RFID se ha sido desarrollado de diferentes maneras por diferentes fabricantes; los estándares internacionales están aún en fase de desarrollo y la interoperabilidad es un problema grave. Cabe señalar que algunos dispositivos RFID nunca han tenido la intención de dejar su red como las etiquetas RFID que se utilizan para control de inventario dentro de una empresa. Esto puede causar problemas para las empresas.

Los clientes también pueden enfrentar problemas con diversos estándares RFID y en los temas de interoperabilidad. Por ejemplo, si alguna compañía es propietaria de alguna etiqueta especializada

de pago rápido, en caso de que otras empresas quieran utilizar esa misma etiqueta convenientemente tendrían que pagar para acceder a éstas, lo cual es probablemente un escenario no deseado. Por otro lado, si cada empresa tiene sus propios sistemas de pago rápido, un consumidor tendría que llevar muchos dispositivos diferentes, lo cual va en contra de la facilidad de uso de la tecnología.

Fácilmente Perturbados

Dado que los sistemas RFID hacen uso del espectro electromagnético (redes WiFi o teléfonos celulares), son relativamente sensibles a fallas y problemas en la energía empleada sobre la frecuencia correcta. A pesar de que esto sólo sería un inconveniente para los consumidores en las tiendas (largas esperas en las cajas registradoras), estos problemas podrían ser desastrosos en otros entornos (por ejemplo, hospitales, campos de batalla) donde RFID es utilizada cada vez más.

Además, las etiquetas RFID activas (las que utilizan una batería para aumentar el rango de alcance del sistema) pueden ser interrogadas en repetidas ocasiones hasta agotar la batería interna, y alterar de esta manera el funcionamiento del sistema.

Colisiones en Lectores

Las colisiones en los lectores o receptores se producen cuando las señales de dos o más lectores se superponen. La etiqueta es incapaz de responder a las preguntas simultáneas. Los sistemas deben ser cuidados al establecer medidas para evitar este problema; muchos sistemas utilizan protocolos anticollisión. Los protocolos anticollisión permiten que las etiquetas se turnen para transmitir hacia un lector.

Colisiones en Etiquetas

Las colisiones en etiquetas se producen cuando estas están presentes en un área pequeña, pero desde que el tiempo de lectura se ha reducido, es fácil para los proveedores desarrollar sistemas que garanticen que las etiquetas respondan una a una a tiempo mediante el empleo de algoritmos adecuados.

4.3.3 Privacidad y Ética

Flexibilidad de Lectura

El contenido de una etiqueta RFID puede ser leído después de que el artículo haya dejado la cadena de suministro. Una etiqueta RFID no puede diferenciar entre un lector y otro. Los lectores RFID son muy portátiles; de esta manera las etiquetas RFID pueden ser leídas desde cierta distancia, de unos pocos centímetros a unos pocos metros.

Esto permite a cualquiera ver el contenido de la etiqueta, incluso cuando se está transitando por la calle. También es posible que algunas etiquetas puedan ser apagadas cuando el artículo salga del alcance establecido.

Etiquetas Difíciles de Remover

Las etiquetas RFID son difíciles de quitar, y algunas son muy pequeñas (menos de medio milímetro cuadrado y tan delgadas como una hoja de papel), otras pueden estar ocultas o embebidas dentro de un producto donde los consumidores no pueden verlas. Las nuevas tecnologías permiten etiquetas RFID impresas justo sobre el producto y no pueden ser removidas.

Etiquetas que Pueden ser Leídas sin Conocimiento

Dado que las etiquetas se pueden leer sin haber sido deslizadas o obviamente escaneadas (como es el caso de bandas magnéticas o los códigos de barras), cualquier persona con un lector RFID puede leer las etiquetas incrustadas en las ropas y otros productos de consumo sin el conocimiento del dueño.

Por ejemplo, es posible que un cliente sea escaneado antes de entrar en una tienda, sólo para ver si está llevando consigo una etiqueta RFID, y así sucesivamente. Es posible entonces que el cliente sea abordado por un empleado que sabe lo que tiene en el bolso, y puede sugerir accesorios u otros elementos que coincidan con la suma que este tiene. Esto puede suponer una grave amenaza para la privacidad y seguridad.

Etiquetas que Pueden ser Leídas a Grandes Distancias

Por diversas razones, los lectores RFID y los sistemas de etiquetas están diseñados de manera que la distancia entre la etiqueta y el lector se mantenga al mínimo. Sin embargo, una antena de gran alcance puede ser usada para leer las etiquetas desde mucho más lejos, esto conduce a problemas de privacidad.

Números Únicos para Etiquetas

En la actualidad, el UPC aplicado con códigos de barras permite a cada producto vendido en una tienda tener un número único que identifica a ese producto. Con los posibles avances en el progreso de un sistema mundial de identificación de productos cada artículo podrá tener su propio número. Cuando el producto es escaneado para la compra y pago, la etiqueta RFID con su número único podrá relacionarse con un número individual de tarjeta de crédito. Esto puede suponer una amenaza para la privacidad y seguridad.

4.3.4 Problemas de Seguridad

Los problemas de seguridad alrededor de la tecnología RFID pueden agruparse dentro de las siguientes categorías:

Propiedad y minería de datos: todos los métodos de recopilación de datos implican privacidad, propiedad de los datos, y el uso ético de técnicas de minería de datos para descubrir las características de un individuo o una organización. Por ejemplo, las tarjetas de fidelidad para clientes podrían utilizarse para averiguar información médica privada acerca de un individuo. Este problema se ha presentado antes de la utilización de la tecnología RFID, pero el volumen de datos proporcionados por las etiquetas RFID añade una nueva urgencia a estos debates.

Robo de datos: en cuanto al robo de datos, hay que considerar dos puntos: tanto el acceso al sistema informático como las técnicas de ataques para robo de datos. Las etiquetas RFID están diseñadas para emitir información y la posibilidad de robo de datos es fácilmente una realidad. Los fabricantes de etiquetas, para contrarrestar esta situación, están añadiendo características de seguridad tales como sistemas criptográficos seguros a las etiquetas y datos.

Forjamiento de datos: la mayoría de las etiquetas RFID son regrabables. Esta característica puede ser bloqueada o activada, según la aplicación y el grado de seguridad. Por ejemplo, en las bibliotecas, las etiquetas RFID son con frecuencia desbloqueadas por conveniencia de los bibliotecarios para la reutilización de las etiquetas en diferentes libros o para realizar un seguimiento de entrada y salida de libros. Pero cuando las etiquetas que deben bloquearse no lo están (por ejemplo, en

las cadenas de suministro), existe la posibilidad de que bromistas o usuarios maliciosos puedan reescribir las etiquetas con información incorrecta o fraudulenta.

4.3.5 Ataques

Una vez conocidos los problemas y algunas situaciones problemáticas que pueden presentarse en sistemas de RFID es importante identificar los posibles ataques que pueden sufrir. Antes de analizar los posibles ataques, es necesario identificar posibles objetivos. Según [?] un objetivo puede ser todo un sistema (si la intención es alterar por completo a un negocio), o puede ser cualquier sección del sistema en general (desde toda una base de datos de inventario de productos a un producto en particular).

Con frecuencia los involucrados en la seguridad de la tecnología de la información tienden a concentrarse únicamente en la protección de los datos. Al evaluar e implementar seguridad en torno a RFID, es importante recordar que algunos activos físicos son más importantes que los datos reales. La base de datos podría nunca ser directamente afectada, aunque la organización podría seguir sufriendo enormes pérdidas.

Consideremos el siguiente ejemplo en el sector minorista [?]. Si una etiqueta RFID fue manipulada de manera que el precio en el punto de venta se redujo de \$200,00 a \$19,95, la tienda sufriría una pérdida del 90 por ciento del precio al menor, pero sin daños a la base de datos del sistema de inventario. La base de datos no es directamente atacada y los datos en la base de datos no han sido modificados o suprimidos y, sin embargo, un fraude fue perpetrado porque parte del sistema RFID ha sido manipulado.

Como se describió anteriormente, el acceso físico está controlado por tarjetas RFID llamadas comúnmente tarjetas de proximidad. Si hay una tarjeta duplicada, la base de datos no se ve afectada. Sin embargo, todo aquel que pasa con la falsificación de la tarjeta recibe el mismo acceso y privilegios que el titular de la tarjeta original.

4.3.6 Manipulación de Radiofrecuencia

Una de las formas más sencillas para atacar a un sistema RFID es evitar que la etiqueta en un objeto sea detectada y leída por un lector. Dado que muchos metales pueden bloquear las señales de radiofrecuencia, todo lo que se necesita para derrotar a un determinado sistema RFID es envolver el objeto en papel de aluminio o ponerlo en una bola metálica recubierta. Esta técnica funciona tan bien que ahora en algunos casos se emiten bolsas protectoras con cada dispositivo de RFID.

Desde el punto de vista de los ataques en el aire, las etiquetas y los lectores son vistos como una sola entidad. A pesar de que realizan funciones contrarias, son esencialmente diferentes caras de una misma porción del sistema de radiofrecuencia. Un ataque a través de la interfaz de aire en las etiquetas y lectores en general cae en uno de los siguientes cuatro tipos de ataques: *spoofing*, inserción, reproducción, y DoS ⁷.

Spoofing

Los ataques de *spoofing* suministran información falsa que parece válida y que el sistema acepta. Típicamente, los ataques de *spoofing* hacen uso de un falso nombre de dominio, direcciones IP, o direcciones MAC. Un ejemplo de *spoofing* en un sistema RFID es una difusión incorrecta de números EPC por el medio cuando un número válido es esperado.

⁷Denial of Service — Denegación de Servicio

Inserción

Los ataques de inserción introducen comandos en vez de datos en el sistema. Estos ataques funcionan porque se parte del supuesto de que los datos están siempre en una determinada zona, y las normales validaciones no se llevan a cabo.

Los ataques de inserción son comunes en sitios Web, donde el código malicioso se inserta en una aplicación web. Un uso típico de este tipo de ataque, es la inserción de comandos y queries SQL sobre una base de datos. Este mismo principio puede aplicarse a una situación RFID, al tener una etiqueta es posible insertar comandos en lugar de datos válidos en el área de almacenamiento de datos.

DoS

Los ataques de DoS, también conocidos como ataques de inundación, tienen lugar cuando una señal es inundada con más datos de los que puede manejar. Estos son bien conocidos porque grandes ataques de DoS han afectado grandes empresas como Microsoft y Yahoo.

Una variación de este bloqueo es Radio jamming, que es bien conocido en el mundo de la radio, y se produce cuando la radiofrecuencia es llenada con una señal ruidosa. En cualquier caso, el resultado es el mismo: se niega la capacidad del sistema de tratar correctamente a los datos entrantes. Variaciones de este caso se pueden utilizar para atacar a los sistemas RFID.

Manipulación de Datos en las Etiqueta

Hasta ahora se han revisado algunos ataques para bloquear la señal de radiofrecuencia que podrían ser utilizados por alguien que desea intentar robar un solo producto. Sin embargo, para alguien que busca robar varios objetos, una manera más eficiente consiste en cambiar los datos en las etiquetas adjuntas a los objetos.

Dependiendo de la naturaleza de la etiqueta, el precio, número de stock, y cualquier otro dato puede ser cambiado. Al cambiar un precio, un ladrón puede obtener un gran descuento, mientras que todavía aparenta comprar el artículo. Otras modificaciones en los datos de una etiqueta pueden permitir a usuarios comprar artículos sujetos a restricciones de edad tales como películas para adultos.

Cuando los artículos con etiquetas modificadas se compran utilizando un sistema automático de registro y chequeo de compra en efectivo, no se pueden detectar los cambios. Sólo es posible con un inventario físico que revele la escasez de un determinado artículo registrado por el sistema. En 2004, Lukas Grunwald mostró un programa que había escrito llamado RF Dump. RF Dump está escrito en el lenguaje Java de Sun y corre en sistemas operativos Debian Linux o Windows XP. El programa escanea las etiquetas RFID a través de un lector conectado al puerto serial de la computadora.

Cuando el lector reconoce una tarjeta, el programa presenta los datos de la tarjeta en una especie de hoja de cálculo. El usuario puede introducir o cambiar los datos y reflejar esos cambios en la etiqueta (ver Figura 4.9). RF Dump también se asegura de que los datos escritos sean de la longitud adecuada para la etiqueta del campo, rellenando con ceros extras, según sea necesario.

Alternativamente, un programa llamado RF Dump - PDA está disponible para su uso en PDAs. RF Dump-PDA está escrito en Perl, y se ejecuta en Pocket PCs bajo el funcionamiento del sistema operativo Linux. Con el uso de un PDA y RF Dump-PDA, un ladrón puede pasar frente a una tienda y cambiar los datos de artículos con la facilidad de utilizar una Pocket PC de mano.

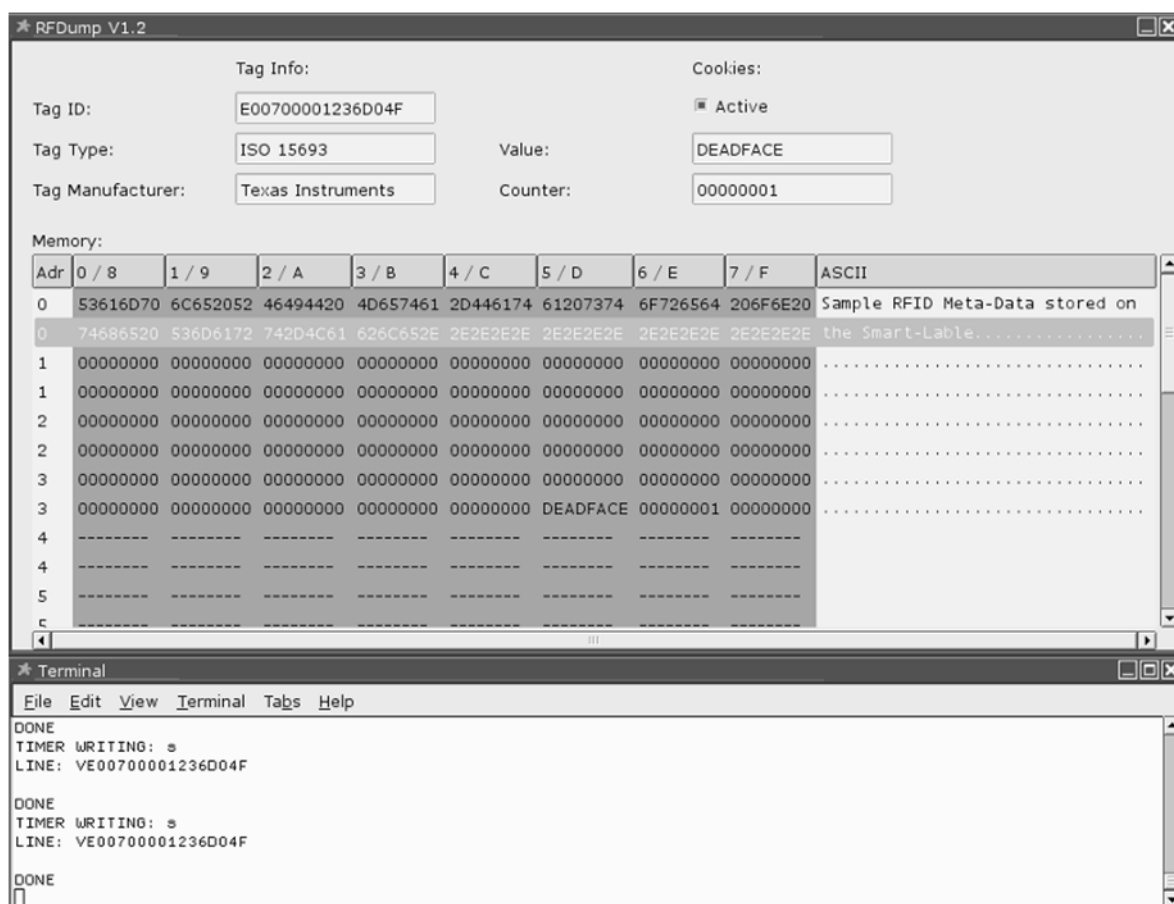


Figura 4.9: RFDump - Cambiando Datos en las Etiquetas. Tomado de [?]

4.3.7 Middleware

Los ataques al *middleware* pueden ocurrir en cualquier punto entre el lector y el *backend*. Veamos un teórico ataque contra el *middleware* de la Exxon Mobil Speedpass⁸.

- La etiqueta cliente RFID Speedpass es activada por el lector. El lector está conectado a la bomba o a un registro de dinero. El lector realiza el *handshakes* con la etiqueta y lee el número de serie cifrado
- El lector y la bomba están conectados a la red de datos de la gasolinera, la que a su vez está conectada a un terminal VSAT de transmisión satelital en la gasolinera
- El VSAT envía el número de serie a un satélite en órbita, el cual a su vez, lo relee y retransmite a una estación satelital terrestre
- Desde la estación satelital terrestre, el número de serie es enviado al centro de datos de Exxon-Mobil. El centro de datos verifica el número de serie y verifica la autorización de la tarjeta de crédito que está vinculada a la cuenta

⁸<http://www.speedpass.com>

- La autorización es enviada de vuelta a la bomba siguiendo la ruta anterior, pero a la inversa
- La caja registradora o la bomba recibe la autorización y permite al cliente hacer sus compras

En cualquier punto del mencionado escenario, el sistema puede ser vulnerado por un atacante externo. Mientras que se requieren sofisticados sistemas transmisores, los ataques contra los sistemas satelitales se han registrados desde la década de 1980.

Sin embargo, el punto más débil en el escenario anterior es probablemente la red de área local. De este recuso se pueden obtener datos válidos para su uso en un ataque, o puede ser la inserción de datos en la LAN, lo que provoque un ataque DoS contra el sistema de pago. Este recurso también podría permitir transmisiones no autorizada.

Otra posibilidad podría ser que una persona técnicamente capacitada tomara un trabajo con el fin de obtener acceso al *middleware*. Estos ataques llamados de ingeniería social tienen lugar cuando alguien toma un empleo con sueldo bajo que permite el acceso al sistema objetivo del ataque.

Mayor cantidad de datos en la transmisión, la conexión entre el satélite y la estación terrestre y el centro de datos donde Speedpass almacena los números de serie, son otros lugares donde el *middleware* puede ser manipulado. Las conexiones entre el centro de datos y la tarjeta de crédito son también puntos en los que el *middleware* puede ser vulnerable.

4.3.8 Backend

Debido a que la base de datos *backend* es a menudo el punto más alejado de la etiqueta RFID, tanto el sentido de los datos y de la distancia física, puede parecer muy alejados como un objetivo para atacar a un sistema RFID. Sin embargo, cabe señalar que seguirán siendo blancos de ataques.

Las bases de datos pueden tener algunos valores intrínsecos si contienen cosas como los números de tarjetas de crédito de clientes. Una base de datos podrá poseer información de gran importancia tales como informes de ventas o secretos comerciales, que es muy valiosa para una empresa competidora.

Las empresas que hayan sufrido daños a sus bases de datos están en riesgo de perder la confianza de los consumidores y, en definitiva, su cuota de mercado, a menos que puedan contener los daños o corregirlos rápidamente. En secciones de negocio de periódicos y revistas han informado de muchas historias en relación con empresas que sufren retrocesos importantes, porque la confianza de los consumidores se redujo debido a un fallo relacionado con tecnologías de información.

La manipulación de bases de datos también puede tener consecuencias el mundo real más allá de la pérdida de los consumidores. Es concebible que las modificaciones de datos en el sistema de inventario de un hospital puede, literalmente, matar personas o cambiar los datos de los pacientes en la base de datos de registros de pacientes puede ser mortal. Un cambio de una letra que implique el tipo de sangre de un paciente podría poner a esa persona en situación de riesgo si recibiera una transfusión. Los hospitales tienen dobles y hasta triples chequeos de control para combatir estos tipos de problemas. Sin embargo, los controles no detendrán cosas malas que ocurran debido a los datos manipulados, sino que sólo pueden mitigar el riesgo.

4.4 Actualidad, Avances y Futuro

El Dr. Alan Kay dijo que "la mejor manera de predecir el futuro es inventarlo". Esto es más cierto ahora que nunca, ya que nuevos competidores y las tecnologías RFID surgen casi a diario, algunas de las cuales son mejoras a los sistemas existentes y algunas de las cuales son significativamente diferentes. Las empresas, en su mayor parte, retienen y esperan el momento propicio antes de decidir qué hacer

con RFID. Las personas que trabajan en los grupos de la EPCglobal e ISO⁹ para forjar las normas están poniendo en práctica la frase del Dr. Kay en el desafío de elegir y definir el futuro en lugar de tratar de predecir el camino que tomará. Esta sección examina las tendencias en las normas, la tecnología y los negocios que están surgiendo de este esfuerzo.

4.4.1 Tecnología

Más inteligentes, más pequeños, más baratos y más rápidos son las recurrentes tendencias en las tecnologías relacionadas con computadoras, por lo que no tiene mucho valor predecir lo mismo para RFID. Pero, ¿qué es lo que probablemente vendrá con el desarrollo de los más inteligentes, más pequeños, más baratos y más rápidos dispositivos RFID? Nuestra expectativa [?] es que esto conducirá a más variados sensores y más etiquetas activas, haciendo los sistemas más manejables y más interconectados. Pero quizás, la predicción más importante es que se producirán algunas *killer app* en los próximos 2 a 4 años que serán tan inesperadas y revolucionarias como la introducción de la World Wide Web en 1990. Vamos a examinar estas predicciones con el fin de tener una idea de lo que el futuro puede traer.

4.4.2 Más Etiquetas Activas

Aunque el reciente incremento del uso de RFID se ha centrado en gran medida en etiquetas pasivas, esto no es probable que siga en caso de que el costo de las etiquetas activas disminuya considerablemente. Los avances en la producción de chips, antenas, y baterías está constantemente reduciendo el costo de las etiquetas activas. Algunos dispositivos EPC incluirán etiquetas clases IV y V y se asume que serán activas.

Un sistema de localización en tiempo real puede utilizar etiquetas de radiofrecuencia, entre otras tecnologías, para llevar a cabo tareas que requerirían un gran número de antenas usando etiquetas pasivas. Por ahora, el costo de las etiquetas activas compensa su enorme flexibilidad, siendo el componente más caro la batería de la etiqueta. Las baterías imprimibles y biodegradables son tecnologías prometedoras ya en uso que pueden reducir en gran medida el costo de las etiquetas activas en un futuro próximo, así como contribuir a revolucionar la manera en que pensamos acerca de la informática en general.

Etiquetas EPC de Clase IV y V

Si bien EPCglobal se ha enfocado recientemente en etiquetas EPC de clase 0, clase I y clase II, todas ellas pasivas, los estándares describen las futuras clases de etiquetas que de hecho serán activas. Etiquetas clase III tendrán una fuente de energía interior para procesadores más rápidos, más memoria, y una función de sensores, pero seguirán siendo pasivas en la transmisión de las comunicaciones. Etiquetas clase IV serán esencialmente para redes inalámbricas *peer-to-peer*. Estas etiquetas no necesitarán de un lector para comunicarse con otras etiquetas de la clase IV. Etiquetas clase V serán lectores en sí mismas, capaces de alimentar y leer etiquetas de Clase I, clase II y clase III, así como comunicarse con otras etiquetas clase IV y clase V.

Sistemas de Localización en Tiempo Real

Dado que una etiqueta activa es capaz de transmitir una señal relativamente fuerte en comparación con la señal reflejada de una etiqueta pasiva, las etiquetas activas pueden utilizarse como localizadores y reconocer espacios en dos o tres dimensiones utilizando la triangulación. Esto significa que con sólo

⁹International Organization for Standardization — Organización Internacional para la Estandarización

dos o tres antenas, un lector puede determinar la ubicación de cada etiqueta en un salón lleno de etiquetas. La Figura 4.10 muestra cómo se podría implementar este sistema. Nótese las dos antenas montadas en las paredes, con sólo estas dos antenas, podemos determinar la ubicación de las etiquetas activas de cada una de los artículos que figuran en los estantes.

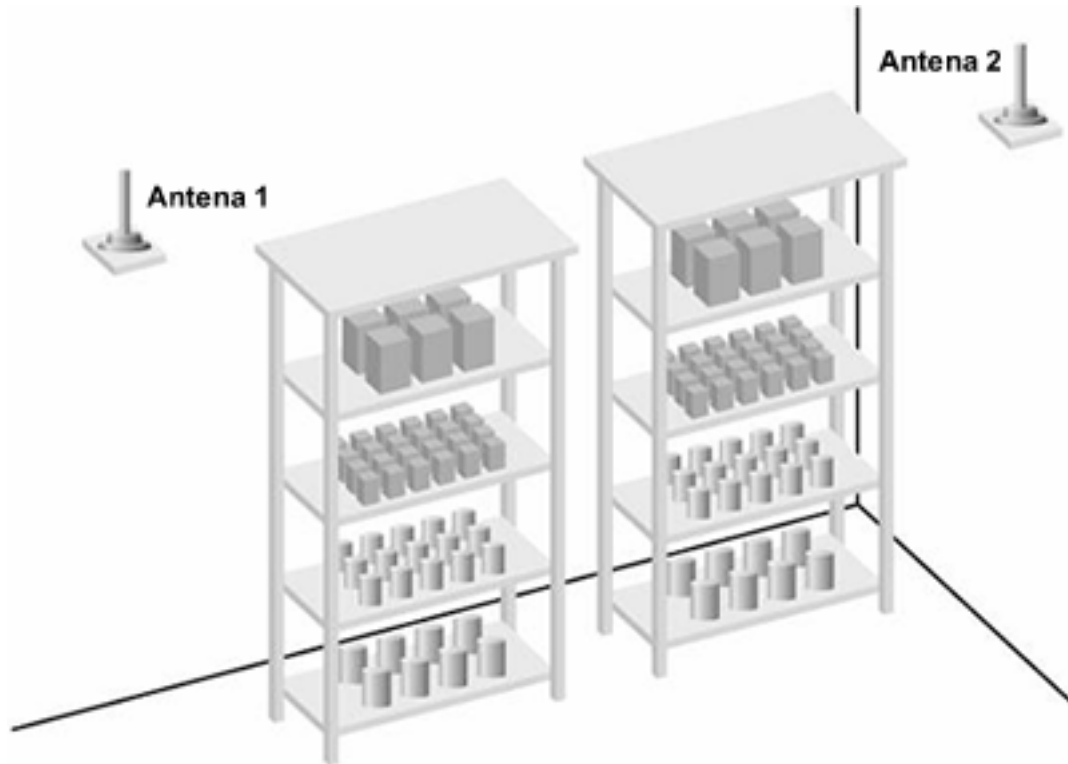


Figura 4.10: Un RTLS en Acción. Tomado de [?]

El uso de etiquetas activas en todo el piso, por ejemplo, del departamento de ventas podría requerir sólo dos antenas para la cobertura. La Figura 4.11 muestra cómo la misma señal de una etiqueta activa llega a las dos antenas en diferentes momentos.

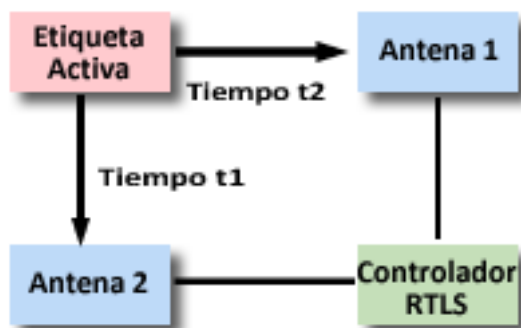


Figura 4.11: Un RTLS en Acción. Tomado de [?]

Al medir con precisión el tiempo en que la señal tarda en viajar desde la etiqueta activa, el controlador es capaz de determinar que el artículo está a cuatro metros de la antena 1 y a dos metros de la antena 2. Esto coloca la etiqueta en un sistema de coordenadas de dos dimensiones.

Baterías Imprimibles

Las impresoras de computadoras son dispositivos muy útiles. Ellas toman algo representado en bits y lo convierten en átomos. Una impresora de cartuchos de tinta sigue a una plantilla en la memoria y a través de un preciso rociado de pequeñas gotas de tinta se transfiere el diseño de bits de la memoria al papel. Ahora bien, si la impresora rocía tinta que contengan zinc, otra tinta que es un electrolito, como un gel de agua salada, y una tercera tinta que contienen dióxido de manganeso, se formará una batería al unir las capas de éstas tintas. La Figura 4.12 muestra las capas.

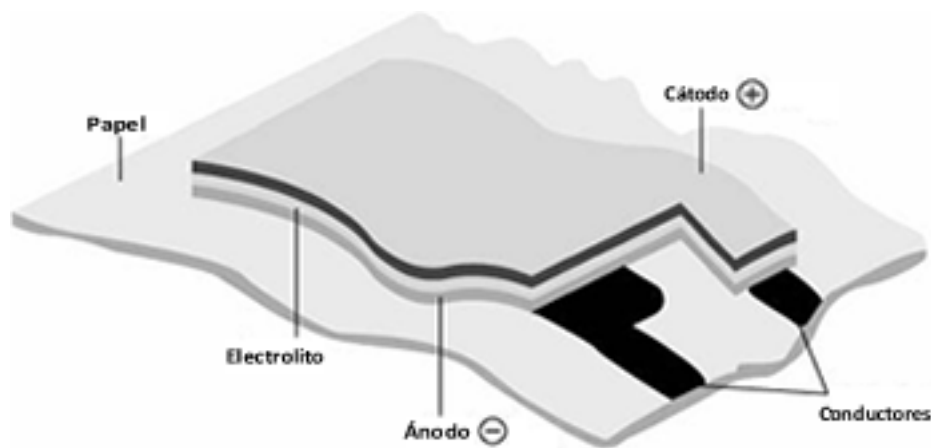


Figura 4.12: Una Batería Impresa en Papel. Tomado de [?]

Estas baterías, ya están en uso para algunos equipos médicos desechables, como los termómetros digitales y son la novedad en material escolar, como carpetas con funciones de calculadoras y juegos. Junto con el delgado, flexible y eficiente circuito impreso en el mismo lugar, se puede convertir algo tan simple como una tarjeta de visita en una aplicación interactiva por sólo una pequeña cantidad extra de dinero.

4.4.3 Más Manejabilidad

La capacidad de administración y extensibilidad son preocupaciones claves para los sistemas RFID. Estas dos propiedades de un sistema son, no casualmente, también las más afectadas por la necesidad de estándares. Con un sistema de numerosos dispositivos individuales ampliamente dispersos en una amplia zona y desplegados en duras condiciones, nuevos dispositivos serán probablemente añadidos además de y en lugar de los viejos dispositivos.

Si la infraestructura de gestión es propietaria, los nuevos tipos de dispositivos pueden ser muy limitados. Incluso un nuevo fabricante para una sonda de temperatura idéntica a la que sustituye podría dar lugar a importantes cambios en el código. Afortunadamente, debido a que estas preocupaciones afectan también a otros tipos de sistemas, como sistemas de manufactura y construcción automatizados, los nuevos estándares serán cada vez más importante para los sistemas RFID en el campo donde estén ya disponibles.

ZigBee

ZigBee es un estándar emergente de capa de red basado en el estándar IEEE 802.15.4. Este simple protocolo puede ser implementado en hardware de bajo costo, requiere muy poco poder, y puede soportar un gran número de sensores y controladores con muy poca latencia. Lo esperado es ver muchos dispositivos con compatibilidad ZigBee en el hogar y en los lectores de RFID en un futuro cercano. Además, las etiquetas RFID activas pueden soportar ZigBee, lo que les permite interoperar con los sistemas de gestión genéricos diseñados para otro tipo de sistemas de automatización, domótica y de manufactura de productos.

Jini

Jini es una tecnología desarrollada por Sun Microsystems que permite a dispositivos descubrirse dinámicamente unos a otros y trabajar juntos en una red. El ejemplo clásico es un sistema Jini para automóviles y teléfonos celulares. El teléfono, al entrar en el carro busca los dispositivos que se describen a sí mismos como altavoces. El teléfono entonces descarga el software de interfaz de un altavoz que permite utilizarlo como un altavoz cuando se recibe una llamada. Cuando el teléfono se retira del carro y es llevado a la casa, la búsqueda de un dispositivo altavoz se inicia en la nueva ubicación.

Esto es agradable como aplicación para un teléfono celular, por ejemplo. Pero es un requisito crítico para una red de sensores. Los sistemas deberán brindar soporte a lo largo del tiempo para nuevos sensores que quizás pueden no haber sido inventados cuando el sistema se diseñó. También se debe considerar sistemas críticos de medio hostil y proveer la capacidad de rutas alternativas frente a fallas. Un sensor puede fallar, lo que requiere que el sistema encuentre una fuente alterna de datos en tiempo real. Jini ofrece las herramientas básicas para este tipo de sistemas dinámicos.

4.4.4 Más Sistemas Interconectados

Todos los sistemas informáticos, no sólo los sistemas RFID, buscan estar cada vez más conectados entre sí, con la Internet y con nosotros. Las mismas tecnologías que soportan esta tendencia en otros sistemas son también importantes para el futuro de RFID. Durante muchos años, el protocolo IPv6 ha sido lento pero inexorablemente extendido en desarrollado y ha ido transportando más y más tráfico en Internet.

Wireless

Un satélite no sería de mucha utilidad si es necesario conectarse a un enchufe de pared, como en los sensores de propagación donde cada vez es más difícil llegar a los rincones de nuestro medio ambiente, es posible encontrar cada vez más dificultades para conectarse a una red cableada. Incluso las redes inalámbricas tradicionales presentan problemas, ya que cada sensor puede no tener una clara línea de comunicación con un hub inalámbrico. Además los repetidores puede ayudar, pero esto significa costo adicional para dispositivos que básicamente no hacen nada más que reenviar paquetes.

Las redes cableadas y eventualmente redes inalámbricas a menudo son configurados de acuerdo a una topología de estrella, anillo o lineal. Es decir, todos los nodos están conectados a un punto central, en caso de una topología de estrella o bien pueden conectarse a uno o dos vecinos para formar un anillo o una topología lineal. Estas topologías son rígidas y fiables, pero frágiles. Una pérdida de un nodo puede ocasionar la caída de toda la red, ya que cada nodo está siempre en un lugar determinado de la topología y sólo se comunica con sus conexiones asignadas.

Una red mallada, por el contrario, es una topología donde cada nodo se conecta a más de un nodo y los paquetes se pasan de un nodo a otro hasta alcanzar el nodo destino. Es extremadamente resistente,

ya que la pérdida de un nodo tiene poco impacto en la red. Las redes de malla también superan los problemas de línea de visión, ya que un nodo puede ser capaz de enviar un paquete a un vecino que luego lo remite a otro que pasa a tener visibilidad del destino final.

Una red de malla parcial inalámbrica. Se trata de una topología donde los nodos se conectan y desconectan de vez en cuando y cada nodo se conecta a algunos, pero no a todos los demás nodos. La Figura 4.13 muestra cómo una red de malla inalámbrica puede “ver en torno a” una obstrucción.



Figura 4.13: Red de malla inalámbrica. Tomado de [?]

La obstrucción de una pared, en este caso bloquea la línea de visión entre los nodos ubicados a la derecha y a la izquierda. Sólo un nodo más allá del muro tiene visibilidad de alguno de los miembros de cada grupo y como la naturaleza de la red es de malla, éste nodo automáticamente transmite la comunicación entre los dos grupos separados. Un sistema RFID compuesto por lectores, etiquetas pasivas, etiquetas activas y otros sensores, podrían ser desplegados como una red de malla, permitiendo que los componentes sean añadidos y quitados con un mínimo de perturbación y reconfiguración.

Realidad Aumentada

Usando un proyector portátil combinado con un lector de RFID, investigadores de Mitsubishi Electric Research Labs han inventado una forma de proyectar las etiquetas directamente sobre objetos identificados por las etiquetas RFID. Esto es más que una nueva interfaz de usuario, esto también implica una idea totalmente nueva para identificar las etiquetas.

Utilizando un lector con un proyector incorporado, un usuario puede encontrar artículos al agitar el lector en alguna dirección. El proyector ajusta el movimiento y la posición y proyecta una marca sobre el o los artículos de interés. El sistema funciona enviando primeramente un pulso de radiofrecuencia que active las etiquetas pasivas. El proyector luego proyecta un patrón sobre el área en general, escaneando toda la zona hasta que el sensor en una etiqueta reconoce el patrón. La etiqueta informa al lector que está actualmente iluminada. El proyector luego proyecta imágenes apropiadas, orientadas en función a cálculos basados en los reportes de las etiquetas iluminadas y, posiblemente, otras informaciones de localización. Ver Figura 4.14.

La imagen proyectada puede ser arbitrariamente compleja, por lo que el dispositivo podría utilizarse para delinear una parte en un estante o para reconocer la presencia o ausencia de componentes específicos. El dispositivo podría mostrar vídeo y gráficos con flechas proyectados en las propias partes para mostrar cómo interpretarlos.

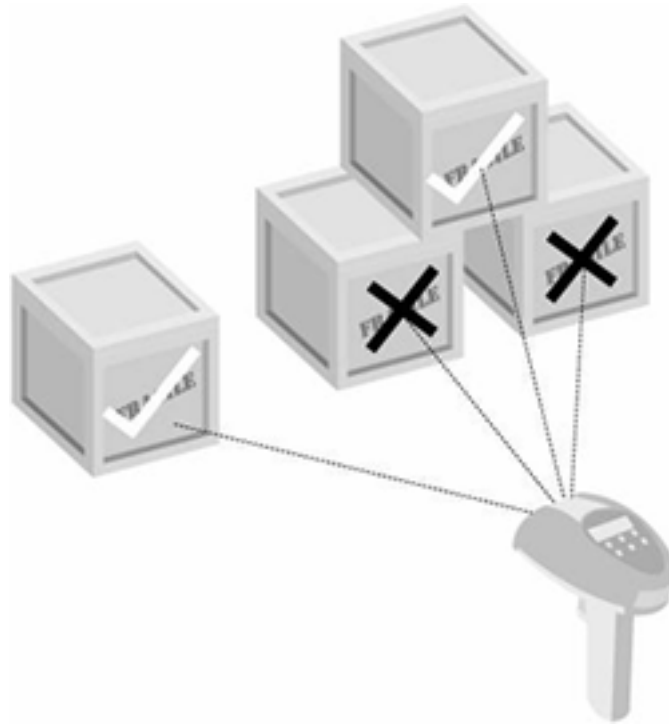


Figura 4.14: RFID facilita realidad aumentada. Tomado de [?]

4.5 Resumen

Los sistemas RFID contemplan importantes aplicaciones y representan mejoras y avances en cuanto a la realización de procesos y funciones en todos los sectores de la vida cotidiana. Por lo tanto, se puede asegurar que esta tecnología tiene su futuro asegurado en empresas públicas y privadas, y muchas instituciones que, de alguna manera, requieran del uso efectivo de tecnologías AIM.

Es por esto que la intención de este capítulo fue la de describir a los sistemas RFID desde el punto de vista sistémico y técnico. Se describieron los componentes esenciales que debe tener todo sistema RFID, así como también las aplicaciones que se les pueden dar a estos sistemas.

Por otra parte, se describieron los aspectos de seguridad que se deben tener en cuenta al momento del desarrollo e implantación de sistemas RFID. Se describieron aspectos que pueden ser considerados controversiales, por representar vulnerabilidades a la privacidad, así como también aspectos positivos que, definitivamente marcan y marcarán el futuro de esta tecnología.

Capítulo 5

El Lenguaje Ruby

Vale la pena recordar que un nuevo lenguaje de programación a veces es visto como una cura para todos los males, especialmente por parte de sus seguidores. Pero un sólo lenguaje no va a suplantarse a todos los demás, porque una sola herramienta no es indiscutiblemente la mejor para cada tarea. Existen muchos tipos de dominios de problemas en el mundo y muchas posibles limitaciones sobre los problemas dentro de los dominios.

En este capítulo, se examinarán las características generales del lenguaje de programación Ruby y algunas de las fortalezas y bondades que ofrece. Igualmente, se examinarán los aspectos más relevantes del sistema de administración de paquetes de Ruby mejor conocido como RubyGems.

5.1 Introducción a Ruby

Ruby [?] es un lenguaje de programación orientado a objetos creado en 1993 por el japonés Yukihiro Matsumoto y su primera aparición para el público fue en 1995. Ruby fue diseñado para mejorar la productividad del programador y para darle un toque de diversión a la programación. Matsumoto hace énfasis en tratar de que Ruby siga el principio de minimizar la confusión de usuarios experimentados, por lo que resulta fácil de aprender lo básico del lenguaje. Es importante destacar que el diseño del lenguaje se enfoca en el humano y no en la máquina.

Ruby es un lenguaje reflexivo, dinámico y tiene un recolector de basura, lo cual da soporte a muchas características interesantes del lenguaje como: tipado dinámico, iteradores, generadores, green threads¹, meta-programación, co-rutinas y continuaciones². Estas características de Ruby se derivan principalmente de Perl, Smalltalk, Python, Lisp, Dylan y CLU.

En Ruby [?], todo lo que se puede manipular es un objeto y los resultados de esas manipulaciones, a su vez, son objetos. Por ejemplo, en Java se puede calcular el valor absoluto de un número llamando a una función aparte y pasándole el número por parámetros:

```
numero = Math.abs(numero); // Código Java
```

En Ruby es posible determinar el valor absoluto de un número gracias a que existe un método disponible en la clase *fixnum* que se encarga de los detalles internamente:

¹hilos de ejecución simultáneos en todas las plataformas

²facilidades de implementación de reflexión computacional

```
numero = numero.abs
```

Lo mismo aplica para todos los objetos Ruby: en C se escribiría `strlen(nombre)`, pero en Ruby sería `nombre.length`. Esto es lo que significa que Ruby sea un lenguaje genuinamente orientado a objetos.

5.1.1 Metaprogramación

Una de las características más interesantes de Ruby es que permite hacer metaprogramación de manera muy fácil y natural, mayormente gracias a que la sintaxis del lenguaje es neutral, discreta y poco complicada. Aunque la metaprogramación es un concepto difícil de explicar, básicamente consiste en escribir código que, a su vez, genera código. La importancia de esta técnica es que permite eliminar duplicación de código y otros aspectos indeseables. El siguiente ejemplo³ es una manera de declarar propiedades de objetos en clases Ruby:

```
attr_reader :id, :age
```

El método `attr_reader` viene incorporado por defecto en Ruby y así es como está definido:

```
1 def attr_reader(*syms)
2   syms.each do |sym|
3     class_eval %{def #{sym}
4       @#{sym}
5     end}
6   end
7 end
```

De esta manera se puede apreciar lo conveniente que puede ser el uso de la metaprogramación en la construcción de código elegante y fácil de entender en Ruby.

5.1.2 El Futuro de Ruby

Un indicador [?] de un gran futuro para un lenguaje de programación es la vitalidad y talento de la comunidad que lo rodea. Siempre se puede estar impresionado por el ingenio y la agudeza de los desarrolladores de Ruby. Es posible encontrar en <http://www.ruby-doc.org/core/> personajes discutiendo temas sobre Ruby: nuevas características de la librería estándar, mejores prácticas, metaprogramación, mejoras de desempeño, entre otros. Es uno de los mejores lugares para obtener información sobre nuevos cambios en Ruby.

Actualmente existen dos proyectos paralelos derivados de la propuesta de Ruby, llamados YARV y JRuby, que básicamente proveen ambientes de ejecución para mejorar y expandir la ejecución de scripts de Ruby.

YARV

Debido a la crítica más común de las aplicaciones Ruby (son más lentas en comparación con otros lenguajes interpretados), existe un equipo de desarrollo encargado de hacer que el intérprete de Ruby sea más rápido. Se planificó tener listo para finales del 2007 un intérprete de Ruby bytecode, llamado Yet Another Ruby VM y que tiene un sólo objetivo: “desarrollar la máquina virtual para Ruby más rápida del mundo”.

³Tomado de: <http://www.vandenburg.org/Speaking/Stuff/oscon05.pdf>

YARV es una máquina virtual simple, basada en pilas, con optimización en tiempo de compilación y compilación *Just In Time*, manejador de hilos nativo (POSIX o Windows), caching de métodos de una línea y muchas características adicionales. En comparación con el intérprete Ruby original, YARV muestra mejoras en programas que presentan problemas clásicos de cuellos de botellas de la máquina virtual o que hacen mucho procesamiento simbólico.

JRuby

JRuby es un intérprete compatible con la versión 1.8.4 de Ruby escrito totalmente en Java, y se avoca a la posibilidad de ejecutar scripts Ruby en cualquier plataforma con soporte para Java. Mientras que el objetivo de YARV es mejorar la velocidad, el de JRuby es máxima portabilidad. De hecho, es posible que el número de plataformas sobre las que Ruby puede ejecutarse sea mayor que Java, a excepción de los dispositivos móviles. Más allá de esto, JRuby ha incorporado mecanismos para definir e interactuar con clases Java desde Ruby. El intérprete también provee soporte para el BSF⁴, permitiendo usar código Ruby en páginas JSP⁵.

JRuby [?] fue creado en 2001 por Jan Arne Petersen y soportaba la semántica de la versión 1.6 de Ruby. El cambio final se presentó hacia los comienzos de 2006 con el objetivo de estar completamente conforme a Ruby. Luego de migrar varias extensiones importantes a Java (YAML, Zlib y otras partes importantes de un sistema Ruby), RubyGems y Rails empezaron a funcionar.

5.2 RubyGems

En pocas palabras, RubyGems [?] permite distribuir e instalar código Ruby en cualquier sistema que ejecute Ruby. RubyGems es un sistema de administración de paquetes para aplicaciones Ruby y librerías. Estos paquetes se denominan *gems* y existe una gran variedad de ellos, cada uno con una especialidad distinta. Para instalar un *gem* determinado basta con ejecutar un simple comando y el sistema puede resolver automáticamente las dependencias e instalar el *gem* deseado y cualquier otro que sea necesario.

5.2.1 ¿Cómo Instalar RubyGems?

El primer paso para instalar RubyGems es instalar el intérprete de Ruby o JRuby. Existen muchas maneras de hacer esto y cada una depende de la plataforma en la que se quiera instalar el intérprete. Un buen sitio para empezar es <http://www.ruby-lang.org>. Una recomendación particular es instalar la versión 6.0 o posterior del IDE⁶ NetBeans, disponible de manera gratuita desde <http://www.netbeans.org>, que viene con soporte para Ruby de manera integrada.

Para instalar Ruby en Windows es muy sencillo:

- Descargar el archivo ejecutable (.exe) desde http://rubyforge.org/frs/?group_id=167. Recomendación: descargar la última versión estable que aparezca en la lista. Por ejemplo, descargar la versión 1.8.6 (**ruby186-26.exe**)
- Ejecutar el archivo y seguir las instrucciones de instalación
- Para probar que se ha instalado correctamente, se puede ejecutar el comando **ruby -v** desde la consola de Windows

⁴Bean Scripting Framework <http://jakarta.apache.org/bsf>

⁵JavaServer Pages

⁶Interface Development Environment — Interfaz de Ambiente de Desarrollo

- Si aparece un mensaje con la versión del intérprete de Ruby, se ha instalado correctamente

Una vez instalado el intérprete de Ruby, basta con descargar el paquete `rubygems-x.x.x.zip`, descomprimirlo y ejecutar el comando `ruby setup.rb`. El paquete puede ser descargado desde http://rubyforge.org/frs/?group_id=126. Para probar que se ha instalado adecuadamente se puede ejecutar el comando `gem help`, el cual desplegará información acerca de los comandos disponibles.

En el caso de haber instalado NetBeans, ya viene instalado por defecto, e incluso tiene algunos *gems* como *Rails*, disponibles para ser usados desde la primera vez.

5.2.2 ¿Cómo Utilizar RubyGems?

Una vez instalado correctamente Ruby y RubyGems, el sistema permite consultar la correcta utilización y las opciones de los subcomandos con `gem <comando> help`. Por ejemplo, para consultar las opciones del comando `install`, se puede ejecutar `gem install help`.

El comando `install`, por ejemplo, permite instalar un paquete y todas sus dependencias especificando el nombre del paquete que se quiere instalar. Por ejemplo: `gem install rails`. Otro comando bastante útil es el comando `gem list`, que lista todos los paquetes que están instalados en el sistema, indicando el nombre y la versión de cada uno.

5.2.3 ¿Cómo Crear un Gem?

Los *gem* se pueden presentar en forma de aplicación o librería. Existen varios aspectos importantes que hay que tener en consideración al momento de generar un *gem*: el *gem* puede depender de otros *gems*, de librerías compartidas (archivos `.so` o `.dll`), de extensiones (archivos `.c` o `.h`, en cuyo caso se deben incluir un archivo de configuración `extconf.rb` con el cual se debe generar un Makefile para compilar la extensión al momento de instalar el *gem*) y, por último, el *gem* puede depender de una plataforma específica (Linux, Mac OS o Windows).

En general, el proceso de generación de un *gem* es muy sencillo. El *gem* debe seguir una estructura particular:

- Los archivos fuente deben estar en una carpeta con el nombre `lib`
- Debe existir al menos un archivo con el nombre del *gem* (p.e. `lib/nombre_del_gem.rb`)
- Debe existir en la raíz del *gem* un archivo con el nombre `README` que contenga una descripción del proyecto, los datos de contacto del autor y una ayuda para utilizar el *gem*
- Los archivos de pruebas deben estar en una carpeta `test`
- Los archivos ejecutables deben estar en una carpeta `bin`
- Si el *gem* contiene extensiones, deben estar en una carpeta `ext`
- Si se tiene documentación sobre el *gem*, ésta debe estar en una carpeta `docs`

El siguiente paso es incluir toda la metadata necesaria en un archivo (comúnmente denominado *gemspec*) para poder luego empaquetar adecuadamente el *gem*. A continuación se muestra un ejemplo sencillo de un *gemspec*:

```

1 # ejemplo.gemspec
2
3 require 'rubygems'
4 SPEC = Gem::Specification.new do |s|
5   s.name           = "ejemplo"
6   s.version        = "1.0.0"
7   s.author         = "Pedro_Perez"
8   s.email          = "pedro@perez.com"
9   s.homepage       = "http://www.pedroperez.com/ejemplo"
10  s.platform        = Gem::Platform::RUBY
11  s.summary         = "Una aplicación de ejemplo"
12  s.require_path    = "lib"
13  s.autorequire     = "ejemplo"
14  s.test_file       = "test/ts_ejemplo.rb"
15 end

```

El gem es un archivo comprimido (como un .zip) que contiene todos los archivos fuente de la aplicación o librería. La manera más sencilla de empaquetar el gem es ejecutando el siguiente comando:

```

1 ruta/de/mi/gem$ gem build ejemplo.gemspec
2 ...
3 Successfully built RubyGem
4 Name: ejemplo
5 Version: 1.0.0
6 File: ejemplo-1.0.0-x86-linux.gem

```

5.2.4 Ventajas

La principal ventaja de usar este sistema es la fácil y rápida reutilización de código. Sin embargo, se pueden destacar otras aplicaciones interesantes que representan mayores ventajas en cuanto a otros sistemas de este tipo:

Provee un mecanismo estándar para describir requerimientos y software: RubyGems permite definir *gemspecs* para describir el software: nombre, versión, descripción, etc., y puede estar incluido en un archivo *.gem*, el cual es un archivo comprimido el cual contiene a su vez, todos los demás archivos requeridos por el software. Este archivo puede ser cargado en RubyForge, lo que permite que pueda ser instalado desde cualquier sistema RubyGems o pueda ser distribuido por los medios tradicionales como HTTP o FTP. Con el comando *gem list* se pueden ver los detalles del *gem* o buscar otros similares.

Provee un repositorio central de software: RubyForge es un repositorio central de software Ruby, sin el cual los programadores tendrían que ubicar, descargar e instalar manualmente un *gem* y todas sus dependencias. Con RubyForge y RubyGems se puede ubicar el software y resolver las dependencias automáticamente.

Permite la distribución de *gems* usando un servidor de *gems*: RubyGems viene con la tecnología necesario para configurar fácilmente un servidor de *gems* en una red local o en Internet. También sirve para hacer cache de todos los *gems* de desarrollo que se usan mayormente para acelerar las descargas o si se prefiere distribuir los *gems* a través de un sitio web personal para no usar RubyForce.

Maneja las dependencias de software automáticamente: cuando se instala un *gem*, el sistema RubyGems es capaz de determinar automáticamente qué otros *gems* son requeridos y pregunta si se quieren instalar. Esto hace mucho más simple la instalación de *gems*, reduciendo significativamente el tiempo que sería necesario para investigar los paquetes requeridos para poner a funcionar adecuadamente el software Ruby.

Maneja diferentes versiones de software inteligentemente: RubyGems puede almacenar múltiples versiones de un software y permite hacer peticiones de paquetes de software de acuerdo a una versión específica, lo cual resulta muy conveniente si se requiere la última versión de un *gem* que contiene unas funciones que no estaban disponibles en versiones anteriores o si la versión más nueva no es compatible.

Puede ser usado transparentemente en lugar de librerías regulares de Ruby: si se instala un software particular desde RubyForge de la manera tradicional o a través de RubyGems, la manera de usar ese paquete es exactamente la misma: con la instrucción `require 'paquete'`, por ejemplo. De esta manera, al distribuir el software, el usuario no requiere tener RubyGems instalado, sino que solamente debe disponer de la librería. Sin embargo, si se requiere una versión específica del software, la solución es usar un comando de RubyGems.

Permite usar la misma tecnología en diferentes sistemas operativos: RubyGems está diseñado para todas las plataformas que ejecutan Ruby. Existen otros sistemas para hacer más fáciles las instalaciones como el `apt`⁷, pero estos dependen del sistema operativo.

5.3 Resumen

En este capítulo se pudieron describir los aspectos más importantes del lenguaje de programación que es considerado por muchos programadores como el lenguaje más divertido y fácil de usar y aprender.

Se describieron características importantes del lenguaje, pero además, se describieron algunos de los proyectos derivados de este lenguaje que hacen que sea aún más interesante y motivante para los programadores. Los proyectos como YARV y JRuby están captando el mayor interés en la comunidad de desarrolladores de software de código abierto.

También se describieron algunos aspectos relacionados con el sistema de distribución de paquetes de software de Ruby, conocido como RubyGems, el cual es una herramienta muy conveniente que facilita la instalación de paquetes de software de manera sencilla y rápida. De esta manera se minimiza el tiempo y el esfuerzo en configurar el ambiente de desarrollo de aplicaciones, porque es posible importar estos paquetes en las aplicaciones y trabajar con ellos como si fueran librerías de código.

⁷Advanced Package Tool de Linux

Capítulo 6

Integración de RFID con Ruby

Cuando se comienza a trabajar para una empresa grande, se tienen muchas expectativas pero, luego de haber pasado tiempo estudiando y analizando el ambiente de programación, es posible darse cuenta que el mundo real luce muy diferente de como se esperaba. Las empresas usan docenas, cientos y algunas veces, miles de aplicaciones, componentes, servicios y bases de datos.

Muchas de ellas, hechas para satisfacer las necesidades específicas, hechas por la misma empresa o por otras, algunas compradas, otras basadas en proyectos de código abierto y algunas otras de origen desconocido. Se ejecutan en sistemas operativos y hardware heterogéneos, usan bases de datos y sistemas de mensajería de diferentes marcas y escritas en diferentes lenguajes de programación.

Las razones de esto son muchas pero, en vez de hacer quejas sobre el esparcimiento de datos valiosos en diferentes esquemas de bases de datos o bases de datos de diferentes marcas, es posible escribir código que integre esa data. Inclusive, es posible escribir nuevas aplicaciones que se integren a los recursos existentes.

No importa si se usan bases de datos relacionales, repositorios LDAP¹, archivos XML o servicios web basados en diferentes protocolos. Se puede mezclar datos desde múltiples y dispersas bases de datos para crear nuevos conocimientos de negocios.

En este sentido, este capítulo introduce los aspectos relacionados a la necesidad de idear mecanismos para la integración de grandes aplicaciones empresariales utilizando el lenguaje de programación Ruby.

De esta manera, el objetivo de este capítulo es exponer el hecho de que existen herramientas que permiten integrar diferentes tecnologías, y Ruby es quizás la más simple y conveniente de utilizar.

6.1 ¿Qué es Software Empresarial?

En el libro [?], Martin Fowler describe: “las aplicaciones empresariales se basan en desplegar, manipular y almacenar grandes cantidades de datos complejos y dar soporte a la automatización de procesos de negocio con esos datos”. Esa es la definición abstracta pero precisa de los que es software empresarial. Los video juegos no hacen más que eso (y los video juegos modernos requieren grandísimas cantidades de datos de gran complejidad). El punto clave en la definición es que las aplicaciones empresariales son usadas para procesos de negocio y no para mostrar naves espaciales.

¹Lightweight Directory Access Protocol — Protocolo Ligero de Acceso a Directorio

Existen muchas otras diferencias entre las aplicaciones empresariales y otros tipos de software. Por ejemplo, las aplicaciones empresariales se, a menudo, creadas sólo para pequeños grupos de usuarios que están en contacto cercano con el equipo de desarrollo, implicando que los desarrolladores conozcan a sus clientes muy bien. En casos extremos, los programas son escritos por una sola persona. Por ejemplo, un generador de reportes especiales para el presidente de la compañía. El software empresarial exige un grupo específico de herramientas. Las grandes cantidades de datos, complejos o no, deben ser almacenados de alguna manera y en algún sitio. A menudo, son almacenados en bases de datos relacionales, pero también pueden estar en archivos de texto plano o repositorios LDAP. Además, el software empresarial moderno se está basando en arquitecturas distribuidas que consisten en muchos componentes de tamaño pequeño o mediano que realizan tareas especializadas y que se interconectan a través de algún tipo de middleware como CORBA, RMI, SOAP o XML-RPC.

Para los desarrolladores de software empresarial les resulta mejor conocer cómo lidiar con tales tecnologías. Tener habilidades para leer datos de una base de datos relacional o acceder a un repositorio LDAP ayuda a concentrarse más en la parte divertida: la aplicación como tal.

6.2 ¿Qué es Integración Empresarial?

Es difícil dar una definición precisa de lo que es integración empresarial por ser un término vago. Simplemente, es cuando se usa un recurso empresarial preexistente para lograr algún resultado. Si se usan una base de datos existente o un servicio web en una aplicación, se está haciendo integración empresarial. Si se contruye un componente nuevo que va a ser usado por otras partes de la arquitectura existente, se está haciendo integración empresarial.

La integración no necesariamente debe ocurrir dentro de una sola empresa. Es posible y muy usual que el software o los datos de dos empresas diferentes deban ser integrados. Si se está usando un sistema de pagos para los clientes, por ejemplo, se está haciendo integración empresarial de software. Existen pocas excepciones al considerar cada actividad de desarrollo en un ambiente de empresa como algún tipo de integración empresarial. La integración empresarial no ocurre cuando se contruyen nuevas partes de software desde cero. En realidad este caso es raro, pero en teoría es una excepción clara.

La integración empresarial a menudo significa integración con software estándar como bases de datos, repositorios LDAP, colas de mensajes, sistemas ERM², entre otros. Si se está usando una de estas tecnologías, existen buenas posibilidades de que se esté haciendo integración empresarial.

6.3 ¿Por Qué Ruby?

La mayoría del software empresarial en producción actualmente ha sido escrito en leguajes como COBOL, C/C++ o Java. Debido a su naturaleza distribuida, el software empresarial hace que las herramientas y los lenguajes de programación sean fáciles de usar. Cuando se tiene que crear una aplicación *stand-alone* pequeña que use una base de datos existente, servicio SOAP o un repositorio LDAP, pareciera no importar si se escribe en C++, Java o Ruby. Pero si se analiza esto más detalladamente, los lenguajes dinámicos como Perl, Python y Ruby ofrecen muchas ventajas, especialmente en ambientes empresariales:

6.3.1 Ventajas de los Lenguajes Dinámicos

- Son interpretados, por lo que no necesitan una fase de compilación y aumenta la velocidad de desarrollo dramáticamente. Después de editar el programa, se pueden apreciar los resultados de

²Environmental Resources Management — Administración de Recursos Ambientales

los cambios inmediatamente.

- El software empresarial está orientado a la manipulación de datos. Los lenguajes dinámicos están diseñados para manejar datos e incluyen tipos de datos de alto nivel como los *hashes*.
- El manejo de memoria es una tarea que se le delega al lenguaje. Esto es una gran ventaja frente a lenguajes como C++ en el que se debe especificar la longitud de cada string que se lee de una base de datos. Los lenguajes dinámicos previenen los desperdicios de memoria, lo cual resulta en un software más conciso, robusto y seguro.
- El software escrito en lenguajes dinámicos se instala como código fuente. De manera que siempre se sabe con exactitud cuál versión se está ejecutando en el sistema de producción. Ya pasaron los días en que se tenía que averiguar cuál ejecutable específico es el adecuado.

Ruby ayuda a realizar muchas tareas más rápidamente, con más elegancia y con más diversión que con cualquier otro lenguaje disponible hoy en día. Pero más importante aún, es necesario conocer las debilidades de Ruby. Ruby es un lenguaje relativamente nuevo, y aunque el núcleo del lenguaje ha madurado lo suficiente y existen muchas librerías excelentes, muchas de las características de Ruby siguen estando incompletas. Las librerías más importantes vienen integradas con todas las distribuciones de Ruby y las distribuciones estándares están creciendo rápidamente en los últimos años.

6.4 Integración con Java

La Máquina Virtual de Java está constantemente evolucionando en una plataforma multilenguaje en respuesta al CLR³ de Microsoft y a la demanda de los desarrolladores que prefieren los lenguajes dinámicos. Existen múltiples implementaciones de lenguajes dinámicos en la Máquina Virtual de Java que incluyen JavaScript, ECMAScript para XML, Python, BeanShell, Groovy y JRuby. De hecho, Java 6 incluye la implementación del Rhino JavaScript de Mozilla.

Integrar lenguajes dinámicos con Java es relativamente simple porque en tiempo de ejecución, todos los lenguajes comparten la misma máquina virtual, lo cual permite que los lenguajes dinámicos puedan hacer llamadas y compartir datos con código Java y viceversa. Varias especificaciones establecen que la integración con lenguajes dinámicos está creciendo importantemente y será más fácil con el tiempo.

Un caso muy interesante de cómo importar código Java desde Ruby se describe en los tutoriales gratuitos que se difunden desde el sitio web del IDE de desarrollo NetBeans. En el enlace⁴ se puede apreciar un video en el que Tor Norbye explica cómo este tipo de integración es posible utilizando NetBeans 6.0, el cual tiene soporte para Java y JRuby. En el video se puede apreciar que es posible importar cualquier clase hecha en Java desde una clase en Ruby con las sentencias `require 'java'` y `import_class 'paquete.ClaseJava'`, donde “paquete” es el nombre de un archivo `.jar`. Luego de esto, se pueden instanciar objetos de la clase Java e incluso hacer llamadas a métodos de esa clase usando, o no, las convenciones de nombres de métodos de Ruby. De manera que simplemente con disponer del intérprete JRuby y una máquina virtual de Java, el proceso de integración es bastante sencillo.

Un ejemplo para ilustrar este proceso sería empezar por crear una clase Java (`Saludar.java`):

```
1 public class Saludar {
2     public Saludar() {
3     }
}
```

³Common Language Runtime — Lenguaje Común de Tiempo de Ejecución

⁴<http://www.netbeans.org/kb/60/ruby/mixed-ruby-java-screencast.html>

```

4
5     public static void main(String args[]) {
6         System.out.println(saludo());
7     }
8
9     public static String saludo() {
10        return "Hola_Mundo_JRuby_desde_Java";
11    }
12 }

```

El siguiente paso es compilar la clase con el comando `javac Saludar.java`. Una vez compilada la clase, se genera el archivo `Saludar.class`. Este archivo se puede empaquetar en un `.jar`⁵ con el comando `jar cvfm holamundo.jar manifest.mf Saludar.class`, donde `manifest.mf` es un archivo que contiene la siguiente declaración: *Main-class: Saludar*. Luego de que se ha empaquetado la clase en un archivo `.jar` se puede instanciar la clase del paquete desde Ruby. Por ejemplo, se puede abrir una consola del interpretador de JRuby con el comando `jirb`. Seguidamente se pueden ejecutar los siguientes comandos:

```

irb(main):001:0> require 'holamundo.jar'
=> true
irb(main):002:0> include_class 'Saludar'
=> ["Saludar"]
irb(main):003:0> Saludar.saludo
=> 'Hola Mundo JRuby desde Java'

```

Así pues, se puede integrar Java con Ruby de manera muy sencilla. Sólo es necesario disponer del intérprete de JRuby. Es importante destacar que existen diferencias en la manera en como se instancian clases Java desde Ruby dependiendo de la versión del intérprete JRuby que se tenga. Es decir, en las primeras versiones se requería escribir la sentencia `require 'java'`, pero en las nuevas versiones ya no es necesario. Sin embargo, estas diferencias son independientes de la plataforma. De manera que el proceso es el mismo si se está en Linux, Windows o Mac OS.

6.5 Extensiones para Ruby

Afortunadamente las extensiones en C o C++ para Ruby no son tan difíciles de crear. Lo más difícil es encontrar las descripciones detalladas para las funciones del API C de Ruby que serán necesarias para ser invocadas desde la extensión. El libro [?] documenta muchas de las extensiones de las funciones de la API, así como la propuesta general para escribir extensiones para Ruby en C.

Una de las herramientas más conocidas de generación de código para la automatización de la integración de código C o C++ con lenguajes dinámicos es el SWIG⁶. El desarrollador crea un archivo de descripción de interfaz, el cual declara funcionalidades en C o C++ para ser exportadas al lenguaje dinámico, con el cual la herramienta genera el código que une las funcionalidades usando las convenciones esperadas por el lenguaje dinámico. Tal código es compilado en una librería que el lenguaje dinámico puede cargar en tiempo de ejecución.

Por otra parte, el módulo `mkmf` de Ruby permite crear extensiones fácilmente. Simplemente se crea un archivo Ruby con el nombre `extconf.rb` con el siguiente contenido:

⁵Java ARchive

⁶Simplified Wrapper and Interface Generator

```

1 require 'mkmf'
2 create_makefile('myextension')

```

donde `myextension` es el nombre del módulo Ruby que se quiere crear (es decir, `myextension.c`). Luego se ejecuta el comando `ruby extconf.rb` para crear un Makefile, el cual contendrá algo parecido a

```

1 gcc -fPIC -I/usr/local/lib/ruby/1.6/i686-linux -g -O2 -o main.o
2 gcc -shared -o Test.so main.o -lc

```

Luego se puede contruir la extensión ejecutando el comando `make`. El resultado de la compilación es el archivo `myextension.so`, el cual puede ser enlazado dinamicamente desde Ruby en tiempo de ejecución con la sentencia `require 'myextension'`. De esta manera se pueden utilizar las funciones definidas en el módulo C desde Ruby.

Se pueden agregar otras directivas al archivo `extconf.rb` para configurar, por ejemplo, rutas o verificar la existencia de ciertas librerías o funciones. Al ejecutar el archivo `extconf.rb` se crean todos los archivos fuentes en C y C++. Al ejecutar el comando `make` se crean las librerías compartidas o librerías enlazadas dinámicamente que Ruby carga en la aplicación cuando se requiere el módulo.

Para un módulo llamado *myextension*, Ruby esperará encontrar una función C con el nombre exacto `Init_myextension` en la librería compartida. Por ejemplo:

```

1 #include "ruby.h"
2
3 void Init_myextension() {
4     /* codigo de la extension */
5 }

```

Una vez cargada la librería compartida, Ruby invoca esta función para inicializar la extensión. Dentro de esta función se llama a las funciones necesarias del API de Ruby para configurar los módulos, clases y funciones Ruby que proveerá en la extensión.

Para un proyecto de integración middleware en Ruby/C, se puede utilizar la propuesta multicapas en la que las aplicaciones Ruby usan un módulo escrito en Ruby. Este módulo envuelve otro módulo escrito en C, el cual accede directamente al sistema middleware. Los beneficios de esta propuesta son:

- La capa de aplicación reside en límites Ruby-a-Ruby y no en Ruby-a-C, lo cual simplifica el desarrollo de la interfaz de la aplicación ya que se puede implementar la mayoría en Ruby
- El módulo Ruby encapsula los límites Ruby-a-C, ocultándolo del acceso directo desde las aplicaciones, lo cual permite utilizar funcionalidades desde Ruby a C y desde C a Ruby, sin afectar la interfaz de la aplicación o forzar errores en las aplicaciones existentes
- Debido a que se oculta el límite Ruby-a-C, el código C no necesita resolver problemas de desfases de impedancia. En cambio, se puede resolver desde la capa C y, en parte, en el módulo Ruby

6.6 Resumen

La integración de software busca resolver problemas de compatibilidad entre múltiples plataformas, mejorar el desarrollo de aplicaciones grandes, aprovechar los beneficios que ofrecen otras tecnologías como lo lenguajes dinámicos, etc. En este capítulo se describieron los aspectos relacionados con algunos mecanismos y métodos de integración disponibles actualmente, en función del lenguaje Ruby.

Parte III

Marco Aplicativo

Capítulo 7

Adaptación de XP y AM

En este capítulo se plantea la metodología y los procesos de desarrollo adoptados durante la implementación del middleware RFID. Para esto, se definen los requerimientos, propuestas de diseño, prácticas y convenciones adoptadas para implementar el sistema. De esta manera, se explica cómo fue adaptado el proceso de desarrollo XP y la metodología AM para la realización del sistema propuesto. A su vez, se explica en qué consiste el sistema y cómo se concibe, con la finalidad de que se puedan comprender mejor los detalles de la propuesta.

Para la implementación del sistema propuesto se ha decidido utilizar el proceso de desarrollo XP y adoptar algunas de las prácticas definidas en la metodología de diseño AM. Esta decisión obedece a la naturaleza propia del sistema que se quiere desarrollar, donde los requerimientos no provienen de un cliente común¹ y se desea que el desarrollo sea flexible y no esté enmarcado en una metodología rígida y definida de manera estricta.

Por otra parte, se desea que el costo asociado al cambio se minimice, lo cual es posible poniendo en práctica las actividades y adoptando los principios definidos por XP y AM. Además, se desea que las entregas del sistema se definan utilizando un esquema flexible en los que se designen responsabilidades, tareas y fechas o tiempos de entrega.

7.1 Iteraciones

Las iteraciones representan intervalos de tiempo en los que se fijan responsabilidades y tareas. Durante el transcurso de cada iteración, se deben implementar las funcionalidades, módulos o modificaciones que sean necesarias para cumplir con la reponsabilidades y tareas fijadas. La duración de cada iteración es de una semana.

7.2 Metas

Los tópicos que surgen al adoptar las propuestas XP y AM incluyen:

1. Planificar tomando en cuenta el costo, el tiempo, el alcance y la calidad
2. Decidir cuántas características o funcionalidades incorporar en cada iteración

¹El cliente en este caso es el tutor, debido a que este trabajo es de investigación



3. Dar prioridades a ciertas iteraciones
4. Designar responsabilidades y prioridades
5. Balancear el riesgo y escoger la duración de las iteraciones
6. Decidir qué hacer cuando no se cumplirán los objetivos de una iteración en el tiempo previsto


Para la realización de este sistema se decidió utilizar las herramientas provistas por el sistema Asamblea (<http://www.asamblea.com>). Se seleccionó la herramienta *metas* para definir las iteraciones del sistema propuesto. Esta herramienta permite definir una responsabilidad y una fecha para la entrega de esa responsabilidad. A una *meta* se le pueden agregar mensajes, tareas y *tickets*. Un mensaje sirve para notificar a los demás desarrolladores alguna idea, dificultad o problema. Las tareas permiten designar actividades a un desarrollador. Los *tickets* sirven para designar una responsabilidad específica a un desarrollador, definiendo una prioridad, horas para cumplir con el *ticket* y el componente del sistema en el que se debe trabajar.

Elaboración de la interfaz de nivel de aplicación del middleware Edit

Elaboración de interfáz o extensión de la clase [ActiveRecord](#) para agregar métodos en modelos, permitiendo el acceso directo al back end de la aplicación (base de datos)

9 days away (Wed, Aug 27) -

 [New Ticket](#)  [New Task](#)  [New Message](#)



Elaboración del wrapper en Linux Edit

Generar el .so definitivo para crear la extensión de Ruby
Hacer las pruebas y adaptar el código para usar las convenciones de Ruby

Figura 7.1: Definiciones de responsabilidades, tareas y fechas utilizando la herramientas de Asamblea para el desarrollo ágil de software

En el url <http://www.asamblea.com/spaces/tesisucv/milestones> se encuentran las iteraciones definidas para la implementación del sistema. En la Figura 7.1 se muestra la definición de dos iteraciones en las que se puede evidenciar el estado de la iteración (en curso, terminada o vencida), una descripción de las responsabilidades y la fecha pautaada para la finalización de la iteración.

Cabe destacar que si los objetivos de una iteración no son cubiertos en el lapso de tiempo previsto, se puede planificar en una iteración posterior la continuación de la iteración incompleta. También es posible definir modificaciones de los objetivos, responsabilidades o tareas de una iteración, ya sea que esté en curso o sea una iteración pendiente. De esta manera se evidencia la flexibilidad de esta metodología.

7.2.1 Actividades

Para el desarrollo de esta propuesta se consideró necesario demarcar en iteraciones este proyecto. De esta manera, el desarrollo se mantiene en concordancia con los principios y valores que define XP, para poder distinguir claramente las fases del proyecto y el producto de software logrado en cada etapa.

Debido a la naturaleza y arquitectura del producto de software propuesto, se decidió adoptar las cuatro actividades identificadas por XP y los principios y prácticas que define AM, con la finalidad de demarcar de una mejor manera las iteraciones. Según y como se definió en el Capítulo 3, estas actividades, principios y prácticas serán adoptadas y adaptadas de la siguiente manera:

Planificación: esta actividad es de suma importancia porque permite identificar las necesidades y proporciona propuestas para solucionar problemas y cubrir esas necesidades. Esta actividad será sustentada por la utilización del sistema de control de versiones *Subversion*. El portal <http://www.assembla.com> provee este servicio de manera gratuita y adicionalmente incorpora herramientas para designar responsabilidades y tareas, que permiten definir claramente requerimientos de sistema, los cuales representan el insumo necesario para llevar a cabo esta actividad. El portal <http://www.rubyforge.org> también provee un servicio gratuito que permite a los desarrolladores de RubyGems alojar sus proyectos auspiciados bajo una determinada licencia.

Diseño: esta actividad podría implicar la elaboración de diagramas de procesos, despliegue o cualquier otro tipo de diagrama o modelo sencillo que ayude a comprender mejor el sistema a medida que se concibe. Es aquí cuando entran en juego las prácticas y principios de AM para lograr la rentabilidad del proyecto y su desarrollo de manera ágil.

Codificación: esta actividad contemplará la implementación de código C y Ruby. El desarrollo será respaldado y controlado usando el sistema de control de version *Subversion*. De esta manera, se mantiene actualizado el código y se sincronizan adecuadamente las versiones dependiendo de la copia del proyecto que esté desarrollando cada programador.

Pruebas: en esta actividad se pueden incorporar pruebas funcionales para comprobar que el comportamiento del API del lector PhidgetRFID-1023 es el esperado. También se incorporarán las pruebas unitarias y de integración que puedan ser importantes para la construcción del gem. La utilización del framework RSpec puede estar presente en la elaboración de una aplicación de ejemplo que utilice el gem. En general, esta actividad consiste en generar código que pruebe que el código de la aplicación se comporte como se espera.

Cabe destacar que se contemplarán iteraciones que no implementen las cuatro actividades antes descritas, debido a que no serán requeridas o podrán adoptarse otras actividades como investigación, refactorización, etc.

7.3 Requerimientos Generales del Sistema

El sistema debe proveer un método simple para implementar un middleware RFID utilizando, en principio, el lector PhidgetRFID. Este sistema estará destinado para ser usado por programadores Ruby que requieran incorporar la tecnología RFID en sus aplicaciones.

El sistema (gem) deberá proveer las funcionalidades básicas de un middleware RFID, pudiendo identificar claramente sus componentes (el adaptador de lectura, el manejador de eventos y la interfaz de nivel de aplicación).

7.3.1 Requerimientos Funcionales

Los requerimientos funcionales del sistema están determinados por cada componente del middleware RFID. Como se explicó en la Sección 4.1.3, las responsabilidades de cada componente dan lugar a los siguientes requerimientos funcionales:

1. Diseñar un adaptador de lectura que provea una capa de abstracción que encapsule la interfáz del lector PhidgetRFID-1023 de manera que no entre en contacto con los desarrolladores de aplicaciones
2. Diseñar un manejador de eventos que provea mecanismos de observación en conjunto, por tiempo y a través de múltiples lectores y que permita filtrar, consolidar, y transformar las lecturas
3. Diseñar una interfáz de nivel de aplicación que provea un mecanismo estandarizado que permita a las aplicaciones registrar y recibir eventos filtrados de un grupo de lectores y que provea un API estándar para configurar, monitorear y administrar el middleware RFID y los lectores y sensores que éste controla

7.3.2 Requerimientos No Funcionales

Las tecnologías seleccionadas para la implementación del sistema permitirán fijar requerimientos adicionales para darle valor agregado al producto final. La idea es ofrecer un sistema que pueda ser de fácil uso y distribución, robusto y escalable. Para esto se pueden fijar los siguientes requerimientos no funcionales:

1. Poner el gem bajo una licencia pública
2. Ofrecer soporte multi-plataforma
3. Propiciar los mecanismos adecuados que el sistema pueda ser escalable

7.4 Aspectos Determinantes del Sistema

El aspecto más importante del sistema es que debe ser un gem para Ruby. De esta manera, se pueden destacar otros aspectos importantes como:

- El gem estará disponible para ser instalado desde RubyForge u otro repositorio de manera gratuita y bajo una licencia libre
- El gem puede ser utilizado y modificado o adaptado por cualquier programador Ruby
- El gem puede ser instalado y desinstalado fácilmente usando RubyGems
- El gem soportará, en principio, el API del lector PhidgetRFID-1023

Estos aspectos, en principio, determinarán otros aspectos también importantes como el soporte multi-plataforma y las posibles funcionalidades adicionales como el soporte para servicio web, generación de reportes, gráficas y estadísticas, entre otras.

Sin embargo, el aspecto más crucial es adaptar el sistema bajo la estructura de un gem de Ruby, ya que esto implica desarrollar el sistema utilizando una estructura específica y dependiente de cada

plataforma. Es decir, construir la extensión de Ruby para manejar el API C/C++ del lector PhidgetRFID para cada plataforma (p.e., .so para Linux, un .dll para Windows o un .bundle para Mac OS). Por otra parte, existe la posibilidad que este gem dependa de otros gems, por lo que se deben agregar las especificaciones pertinentes. Otro aspecto importante es la adecuada documentación del gem, ya que es uno de los requisitos para poder importar el gem en el repositorio RubyForge, así como también, agregar los tests necesarios para garantizar el correcto funcionamiento del gem.

Otras condiciones aplican para la elaboración del gem. Sin embargo, la propuesta contempla, como mínimo, la extensión de Ruby para el API C/C++ del lector PhidgetRFID y las funcionalidades básicas de un middleware RFID agrupadas en un gem de Ruby.

7.5 Metáfora del Sistema

A través de una metáfora se pueden describir funcionalidades del sistema utilizando diagramas, bosquejos en lápiz y papel o simplemente frases que ayuden a comprender aspectos objetivos del sistema o la lógica que éste implemente.

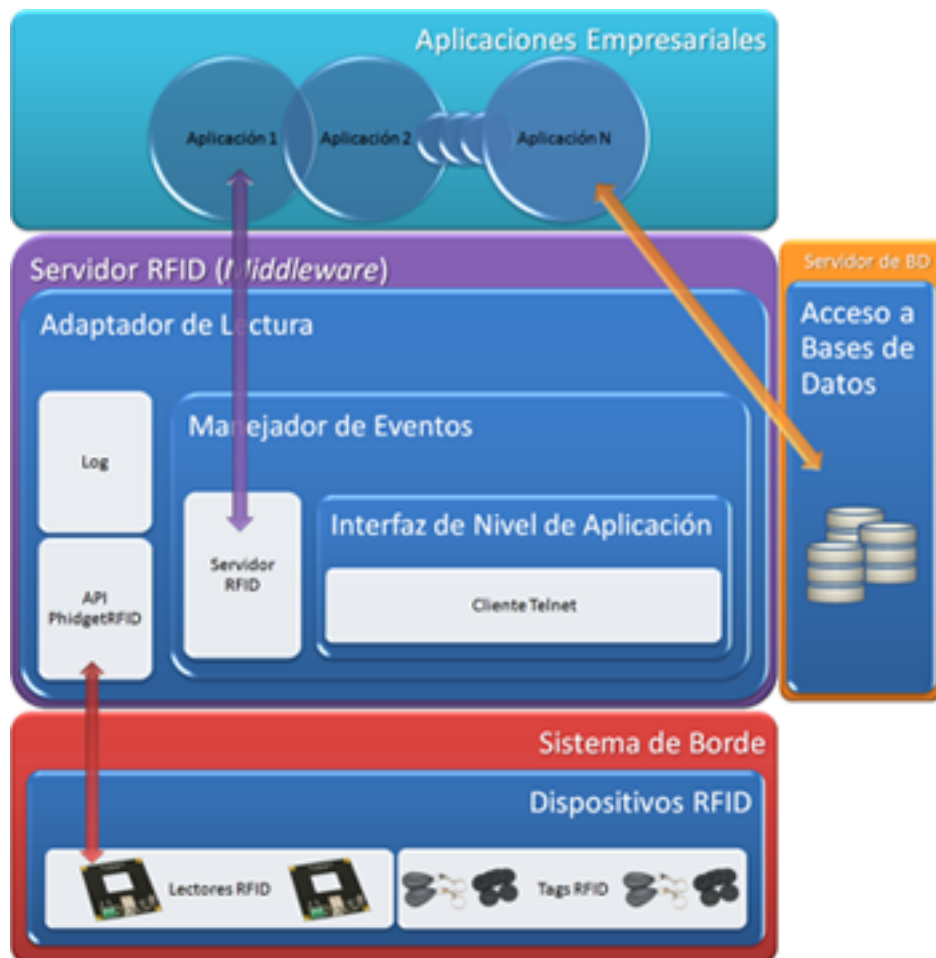


Figura 7.2: Arquitectura del gem de Extensión de Ruby para Sistemas RFID

En la Figura 7.2 se puede apreciar una arquitectura genérica de un posible sistema RFID contruido

usando el gem propuesto. Partiendo de la configuración del middleware RFID se pueden identificar aisladamente los componentes del sistema RFID completo. El servidor RFID estaría encargado de procesar peticiones y respuestas de los clientes (en este caso, las aplicaciones empresariales). El servidor RFID además debe encargarse de manejar todos los eventos asociados al sistema de borde (p.e., un nuevo lector conectado al sistema, un tag RFID leído, etc.). Finalmente, dependiendo de las posibles peticiones que se puedan presentar por parte de las aplicaciones, el servidor RFID debe estar en la capacidad de comunicarse con otros servidores RFID o con servidores de bases de datos para recolectar información relacionada con las etiquetas RFID.



Figura 7.3: Un Posible Escenario de Uso del gem en una Aplicación RFID

Por otra parte, la Figura 7.3 muestra un posible escenario en el que se evidencia que la aplicación cliente no necesariamente necesita estar ejecutándose en la misma máquina donde se encuentra conectado el lector RFID. Y, como se explicó anteriormente, también es posible que el sistema contemple múltiples servidores RFID, lo cual introduce la necesidad de comunicar a los distintos middlewares configurados en cada servidor.

Capítulo 8

Desarrollo

En este capítulo se exponen las actividades de desarrollo e implementación de cada iteración, siguiendo el esquema propuesto en el Capítulo 7, documentando las actividades de diseño e implementación y las estrategias de pruebas.

8.1 Iteración 0

Del 01-Ago-2008 al 08-Ago-2008

En esta iteración se desarrolló el wrapper del API C del lector PhidgetRFID-1023 para poder usarlo desde Ruby.

8.1.1 Planificación

Para esta iteración se definió la meta “[Elaboración del wrapper en Linux](#)”. Los objetivos fijados para esta iteración incluyen:

- Generar un .so (extensión de Ruby)
- Hacer las pruebas y adaptar el código para usar las convenciones de Ruby

8.1.2 Diseño

El problema que se plantea es principalmente la adaptación del API C del lector PhidgetRFID-1023, de tal manera que, a través del generador de interfaces SWIG, se genere un wrapper que permita conectar dicho API C con el lenguaje Ruby. Luego se genera un Makefile utilizando el módulo ‘mkmf’, como se explicó en la Sección 6.5 para poder compilar el wrapper y generar un archivo .so¹, el cual provee los métodos definidos en el wrapper para poder ser usados desde Ruby.

8.1.3 Codificación

El siguiente fragmento muestra cómo se adaptó el API C del lector para disponer de los principales métodos que permiten leer etiquetas y obtener los datos (nombre del dispositivo, número de serial, etc.) de un lector conectado a algún puerto USB del computador:

¹Shared Object — Objeto o Librería Compartida

```

1 // file: phid.h
2 #include <phidget21.h>
3 CPhidgetRFIDHandle rfid = 0;
4 ...
5 const char* get_device_name() {
6     const char *name;
7     CPhidget_getDeviceName((CPhidgetHandle) rfid, &name);
8     return name;
9 }
10 ...
11 void turn_antenna_on() {
12     CPhidgetRFID_setAntennaOn(rfid, 1);
13     CPhidgetRFID_set_OnTag_Handler(rfid, tag_handler, NULL);
14     CPhidgetRFID_set_OnTagLost_Handler(rfid, tag_lost_handler, NULL);
15 }
16 ...

```

El siguiente paso en la elaboración del wrapper consiste en generar el código C adaptado a las reglas que define Ruby. Para esto se utilizó el generador de interfaces SWIG, de tal manera que se definió un módulo *Phid*, el cual incluye los métodos definidos en el wrapper. A continuación, se describe paso por paso el proceso de generación del wrapper:

1. Se agregan las siguientes directivas en un archivo .i (p.e., phid.i)

```

1 // file: phid.i
2 %module phid
3 %{
4 #include <phidget21.h>
5 #include <phid.h>
6 %}
7 %include <phid.h>

```

2. Se genera el wrapper con el comando `swig -ruby phid.i`
3. Se genera un archivo `extconf.rb` con el siguiente contenido (no necesariamente se debe llamar `extconf.rb`)

```

1 # file: extconf.rb
2 require 'mkmf'
3 $LIBS += "_lphidget21"
4 create_makefile('phid')

```

4. Se ejecuta el archivo `extconf.rb` (a través del comando `ruby extconf.rb`) el cual generará un archivo `Makefile`
5. Se compila el wrapper a través del comando `make`, el cual generará una librería compartida (`phid.so`) que puede ser usada desde Ruby utilizando el comando `require 'phid'`

Una vez que se carga la librería compartida (el archivo `.so`), el intérprete de Ruby permite utilizar los métodos definidos en esa librería. Por ejemplo, en el caso del wrapper del lector `PhidgetRFID`, se definieron, entre otros, los métodos `get_device_name` y `turn_antenna_on`, dentro de un módulo de Ruby llamado `Phid`.

8.1.4 Pruebas

Las pruebas se hicieron bajo el siguiente ambiente:

Sistema Operativo

Ubuntu 8.04 i686 GNU/Linux

Configuración de Hardware

CPU: Pentium D 1.48Ghz

Memoria RAM: 1GB

Disco Duro: 120GB

Software

- SWIG versión 1.3.33
- GNU make
- GNU gcc
- Intérprete Ruby versión 1.8.6

Pruebas Funcionales

Para asegurar el correcto funcionamiento de la interfaz que proveen los métodos del wrapper entre el lector y un programa escrito en Ruby, se implementa un programa que muestre por la salida estándar el resultado de la llamada a cada método.

```

1 # file : ts_phid.rb
2 require 'phid'
3 p = Phid
4 p.phidget_create
5 p.phidget_open
6 puts p.get_device_name
7 puts p.get_serial_number
8 p.turn_antenna_on
9 p.phidget_close
10 p.phidget_delete

```

La ejecución del programa anterior muestra el siguiente resultado:

```

1 \$ ruby ts_phid.rb
2 Phidget-RFID 4 output
3 55841
4 Tag Read: 44a322e291
5 Tag Lost: 44a322e291

```

De esta manera se puede evidenciar que se produce el resultado esperado, al mostrar el nombre del lector (`puts p.get_device_name`), el serial del lector (`puts p.get_serial_number`), y el resultado de un evento de lectura de la etiqueta *44a322e291*.

8.2 Iteración 1

Del 08-Ago-2008 al 15-Ago-2008

En esta iteración se agregaron más funcionalidades al wrapper, se definió una clase de Ruby para manejar las funciones del lector a través de una instancia particular.

8.2.1 Planificación

Para esta iteración se definió la meta “Generación de la primera capa del middleware (gem)”. El objetivo fijado para esta iteración incluye:

- Elaboración del adaptador de lectura del middleware RFID

8.2.2 Diseño

Ya que se puede disponer de las funcionalidades principales del lector desde Ruby, la idea ahora es proveer un “adaptador” estándar que permita utilizar la interfáz de un lector genérico. Para ello se implementó el patrón de diseño *Template Method*. La solución consiste en implementar una clase “abstracta” *RfidReader*, de la cual pueden heredar tantas clases concretas como APIs de lectores se tengan. Sin embargo, como la propuesta sólo contemplará un sólo tipo de lector, la única clase concreta que se desarrolló fue la clase *PhidgetRfidReader*.

8.2.3 Codificación

El siguiente fragmento muestra cómo se diseñó la clase *PhidgetRfidReader*. Un aspecto importante que hay que tomar en cuenta es que Ruby permite importar en una clase los métodos de un módulo, con lo cual se contruye un *mixin*. Esta es una manera muy conveniente de utilizar el wrapper del lector recientemente generado:

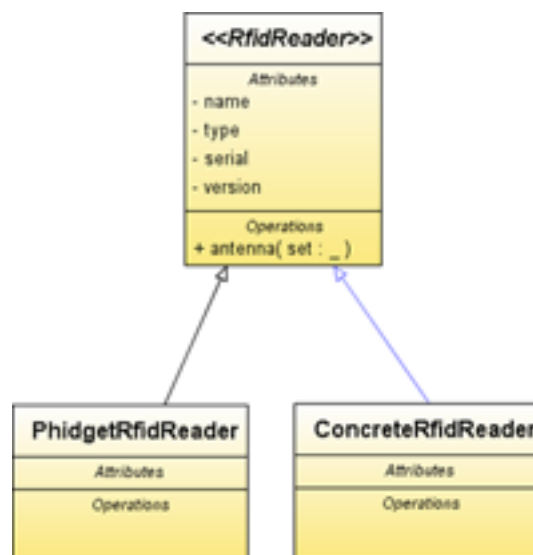


Figura 8.1: Clases concretas y la clase abstracta *RfidReader*

De esta manera la clase abstracta contendrá lo siguiente:

```

1 # file: rfid_reader.rb
2 class RfidReader
3   attr_accessor :name, :type, :serial, :version
4   def initialize(name, type, serial, version)
5     @name=name
6     @type=type
7     @serial=serial
8     @version=version
9   end
10 end

```

Por otra parte, la clase concreta presentará el siguiente contenido:

```

1 # file: phidget_rfid_reader.rb
2 require 'phid'
3 class PhidgetRfidReader < RfidReader
4   include Phid
5   def initialize
6     rfid_create
7     wait_attachment(0)
8     super(get_device_name, get_device_type, get_serial_number,
9           get_device_version)
9     turn_antenna_on
10  end
11  ...
12  def delete
13    rfid_delete
14  end
15 end

```

8.2.4 Pruebas

Las pruebas de esta iteración se hicieron bajo el mismo ambiente descrito en la iteración 0.

Una manera simple de probar el adaptador de lectura para el dispositivo PhidgetRFID es crear un programa sencillo en Ruby que muestre por consola los eventos de lectura de una etiqueta RFID.

```

1 # file: ts_phidget_rfid_reader.rb
2 require 'phidget_rfid_reader'
3 p = PhidgetRfidReader.new
4 while true
5   cmd = gets
6   if cmd == 'close\n'
7     p.delete
8     break
9   end
10 end

```

La ejecución del programa anterior muestra el siguiente resultado:

```

1 \$ ruby ts_phidget_rfid_reader.rb
2 Tag Read: 44a322e291

```

```

3 Tag Lost: 44a322e291
4 close

```

El programa instancia un objeto de la clase *PhidgetRfidReader*. En Ruby, cuando se instancia un objeto de una clase (se llama al método **new**), se ejecuta el método **initialize**. En este método se hace una llamada al método **wait_attachment** definido en el módulo *Phid*, el cual recibe un parámetro que indica el tiempo en milisegundos a esperar para que se conecte el lector (0 para esperar indeterminadamente). Cuando el lector es conectado al puerto USB, continúa la ejecución normal del programa. En este caso se ejecuta el método **turn_antenna_on** el cual habilita la antena del lector y asigna los manejadores de eventos de lectura de etiquetas. A continuación se muestra una sección del código del wrapper que implementa los manejadores de estos eventos:

```

1 // file: phid.h
2 ...
3 int tag_handler(CPhidgetRFIDHandle RFID, void *usrptr, unsigned char *
   TagVal) {
4     turn_led_on();
5     printf("Tag_Read:%02x%02x%02x%02x%02x\n", TagVal[0], TagVal[1], TagVal
       [2], TagVal[3], TagVal[4]);
6     return 0;
7 }
8 ...
9 void turn_antenna_on() {
10     CPhidgetRFID_setAntennaOn(rfid, 1);
11     CPhidgetRFID_set_OnTag_Handler(rfid, tag_handler, NULL);
12     CPhidgetRFID_set_OnTagLost_Handler(rfid, tag_lost_handler, NULL);
13 }
14 ...

```

Los eventos que se activan cuando se lee una etiqueta y cuando la etiqueta sale del rango de lectura del lector imprimen por STDOUT (salida estándar) la cadena **'Tag Read: <id>'** o **'Tag Lost: <id>'**. Finalmente, el programa de prueba termina cuando se recibe por STDIN (entrada estándar) la cadena **'close\n'**.

8.3 Iteración 2

Del 15-Ago-2008 al 22-Ago-2008

En esta iteración se definieron las especificaciones estructurales del wrapper. Con la finalidad de proveer una interfaz adecuada, se implementó, utilizando una base de datos *sqlite*², un repositorio de eventos de lecturas. De esta manera, las aplicaciones que utilizan el wrapper se pueden enterar de los eventos que suceden en el sistema de borde.

8.3.1 Planificación

Para esta iteración se continuaron las actividades de diseño, codificación y pruebas para la meta “Generación de la primera capa del middleware (gem)”, la cual fue definida en la iteración 1.

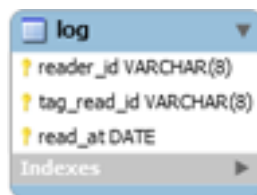
8.3.2 Diseño

La disposición de una base de datos sencilla como *sqlite* provee un mecanismo simple para hacer consultas sobre eventos de lectura ocurridos en el sistema de borde. La manera de implementar este mecanismo consiste en agregar, a nivel del wrapper, al menos dos métodos importantes:

- Un método para crear la base de datos al momento de crear un manejador para el lector
- Un método para registrar un evento de lectura

La razón por la que se decidió implementar este mecanismo usando *sqlite* es que esta base de datos tiene soporte multiplataforma, lo cual la hace portable y muy sencilla de utilizar. El manejador *sqlite* almacena los datos en el sistema de archivos del computador. Es decir, que puede ser copiado y movido usando los mecanismos convencionales del sistema operativo. Esto es muy conveniente ya que permite una mayor libertad y flexibilidad al momento de acceder al historial de eventos del sistema.

De manera opcional, se puede implementar un mecanismo similar utilizando un archivo log en el cual se almacenan los eventos utilizando texto plano. La estructura de la tabla *log* es la siguiente:



log	
reader_id	VARCHAR(8)
tag_read_id	VARCHAR(8)
read_at	DATE
Indexes	

Figura 8.2: Estructura de la tabla *log*

8.3.3 Codificación

El siguiente fragmento muestra cómo se implementaron los métodos básicos necesarios en la implementación del historial de eventos utilizando la base de datos *sqlite*:

²SQLite es una librería que implementa un motor de base de datos SQL transaccional, que no necesita ser instalado o configurado, opera directamente a nivel del sistema de archivos y no requiere de librerías externas o del sistema operativo

```

1 # file: phid.h
2 ...
3 int tag_handler(CPhidgetRFIDHandle RFID, void *usrptr, unsigned char *
    TagVal) {
4     ...
5     sprintf(sql, "INSERT INTO log VALUES('%d', '%02x%02x%02x%02x%02x',
        DATETIME('now'))", get_serial_number(), TagVal[0], TagVal[1],
        TagVal[2], TagVal[3], TagVal[4]);
6     ...
7     rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);
8     ...
9 }
10 ...
11 void rfid_create(char *dbname) {
12     ...
13     rc = sqlite3_open(dbname, &db);
14     rc = sqlite3_exec(db, "CREATE TABLE IF NOT EXISTS log (reader_id TEXT_
        NOT_NULL, tag_read_id TEXT NOT_NULL, read_at DATETIME NOT_NULL,
        PRIMARY KEY (reader_id, tag_read_id, read_at));", callback, 0, &
        zErrMsg);
15 }
16 ...

```

Como se puede apreciar en el método *rfid_create*, se crea una base de datos con el nombre que se reciba por parámetros y se crea una tabla con el nombre 'log'. Esta tabla, simplemente contiene un campo para registrar el identificador del lector, otro campo para el identificador de la etiqueta leída y la fecha y hora del evento de lectura registrado.

Para lograr el correcto funcionamiento del wrapper modificado, es necesario agregar cambios en la clase *PhidgetRfidReader*:

```

1 # file: phidget_rfid_reader.rb
2 require 'phid'
3 require 'rfid_reader'
4 class PhidgetRfidReader < RfidReader
5     include Phid
6     def initialize(log='phid.log', dbname='rfid_events.db')
7         begin
8             ...
9             log_message("#{self}", "Device_#{name}::#{serial}_Found!\n")
10            enable_loggin("#{serial}-#{log}")
11            ...
12        end
13    end
14    ...
15    def delete
16        ...
17        log_message("#{self}", "Device_#{name}::#{serial}_Dettached!\n")
18    end
19 end

```

El método *rfid_create* ahora recibe un parámetro, el cual indica el nombre de la base de datos que se va a crear para el historial de eventos de lectura. Si se instancia la clase *PhidgetRfidReader* sin especificar el parámetro *log* o el parámetro *dbname*, se aceptarán los valores por defecto 'phid.log' y 'rfid_events.db' respectivamente. Esto quiere decir que el nombre de la base de datos se creará en el sistema de archivos, en la ruta en la que se esté ejecutando la clase *PhidgetRfidReader* y con el nombre respectivo. Por otro lado se implementó, usando los métodos *enable_login*, *disable_login* y *log_message* disponibles desde el wrapper, un log de eventos en un archivo de texto plano.

8.3.4 Pruebas

Las pruebas de esta iteración se hicieron bajo el mismo ambiente descrito en la iteración 0.

Pruebas Funcionales

El mecanismo utilizado para probar la funcionalidad del historial de eventos en la base de datos *sqlite* consiste en instanciar el programa de prueba utilizado en la iteración anterior (iteración 1):

```

1 # file : ts_phidget_rfid_reader.rb
2 require 'phidget_rfid_reader'
3 p = PhidgetRfidReader.new
4 while true
5   cmd = gets
6   if cmd == 'close\n'
7     p.delete
8     break
9   end
10 end
```

La ejecución del programa anterior muestra el mismo resultado que se pudo apreciar en las pruebas de la iteración 1:

```

1 \$ ruby ts_phidget_rfid_reader.rb
2 Tag Read: 44a322e291
3 Tag Lost: 44a322e291
4 close
```

Sin embargo, para probar que en efecto se registró el evento de lectura en la base de datos, es necesario hacer una consulta a la misma. Esto se puede hacer desde Ruby utilizando el gem *sqlite3-ruby*. Una vez instalado el gem *sqlite3-ruby*, se escribió un programa sencillo para hacer una consulta a la base de datos generada:

```

1 # file : ts_query_sqlite.rb
2 require 'rubygems'
3 require 'sqlite'
4 db=SQLite3::Database.new('rfid_events.db')
5 cmd='select *_from_log'
6 db.execute(cmd) do |row|
7   row.each do |col|
8     print "#{col.to_s}; "
9   end
10  print "\n"
11 end
```

El resultado de ejecutar el programa anterior muestra los registros que se generaron cuando el lector capturó la etiqueta *44a322e291*:

```
1 \ $ ruby ts_query_sqlite.rb
2 55841;44a322e291;2008-08-21 17:47:29;
3 ...
```

Ahora los eventos que se activan cuando se lee una etiqueta imprimen por STDOUT (salida estándar) la cadena 'Tag Read: <tag_id>', guardan el evento en un archivo log y en una base de datos *sqlite*.

Finalmente, la correcta funcionalidad del archivo log se puede comprobar simplemente al abrir con cualquier editor de texto, el archivo '.log' generado. Por ejemplo:

```
1 ...
2 Thu Ago 21 17:47:27 2008,-1210390864,"clog.c(46)",INFO,"Enabling_logging"
3 Thu Ago 21 17:47:27 2008,-1210390864,"#<PhidgetRfidReader:0xb7c6f590>",
  VERBOSE,"Device_Phidget_RFID_4-output::55841_Found!"
4 Thu Ago 21 17:47:29 2008,-1237988464,"Reader_Serial:55841",VERBOSE,"Tag_
  Read:44a322e291"
5 Thu Ago 21 17:47:29 2008,-1221203056,"Reader_Serial:55841",VERBOSE,"Tag_
  Lost:44a322e291"
6 Thu Ago 21 17:47:42 2008,-1210390864,"#<PhidgetRfidReader:0xb7c6f590>",
  VERBOSE,"Device_Phidget_RFID_4-output::55841_Dettached!"
7 ...
```


8.4 Iteración 3

Del 22-Ago-2008 al 29-Ago-2008

En esta iteración se definieron los componentes finales del wrapper, la estructura y las funciones que debe presentar, con la finalidad de delimitar claramente las funcionalidades del middleware.

8.4.1 Planificación

Para esta iteración se siguió trabajando sobre la meta definida en la iteración 1: “[Generación de la primera capa del middleware \(gem\)](#)”. Los objetivos fijados incluyen:

- Especificar las funcionalidades del wrapper
- Especificar los métodos de uso desde Ruby del wrapper
- Delimitar las funcionalidades del wrapper para diferenciar, desde el punto de vista “físico”, los componentes del adaptador de lectura, el sistema de borde, y las demás capas y componentes del gem.

8.4.2 Diseño

Para entender mejor la estructura del sistema planteado, es importante delimitar el alcance del wrapper, ya que es la “pieza” más importante y sobre la cual se basan las demás funcionalidades del gem. En la Figura 8.3 se pueden apreciar los componentes principales que constituyen el wrapper, y los insumos y procesos necesarios para generarlo.

De esta manera, el wrapper estaría compuesto por tres elementos:

1. La extensión (librería compartida) de Ruby *phid.so*, la cual permite utilizar las funcionalidades principales para manipular el lector (crear y eliminar el manejador para el lector, activar o desactivar la antena y el LED del lector, definir el nombre de la base de datos *sqlite* y el archivo log, entre otras funcionalidades importantes)
2. La base de datos *sqlite*, la cual permite acceder al historial de eventos de lectura. Es decir, el registro de interrogaciones hechas por un lector, que resultaron en lecturas efectivas de alguna etiqueta.
3. El log de eventos asociados a un lector en específico, a través del cual se pueden conocer eventos como el instante en que se conectó o desconectó un lector a un puerto USB del computador, el instante en que se leyó una etiqueta, errores, entre otros.

Finalmente, es importante destacar que este diseño permite un único punto de acceso para consultas al sistema de borde, el cual es la base de datos. Ya que es el único elemento del wrapper que puede ser consultado por un usuario para, por ejemplo, conocer qué etiquetas se han leído dentro de un período determinado de tiempo, o las lecturas hechas por un lector determinado, etc.

De esta manera, se simplifica el acceso para consultas al sistema de borde porque las funcionalidades críticas como la concurrencia son manejadas por el mismo manejador de la base de datos *sqlite*.

Una vez delimitadas las funcionalidades del wrapper, es posible definir una metodología estándar para manipular y acceder a tales funcionalidades desde las capas del middleware, las cuales se contruyen en Ruby.

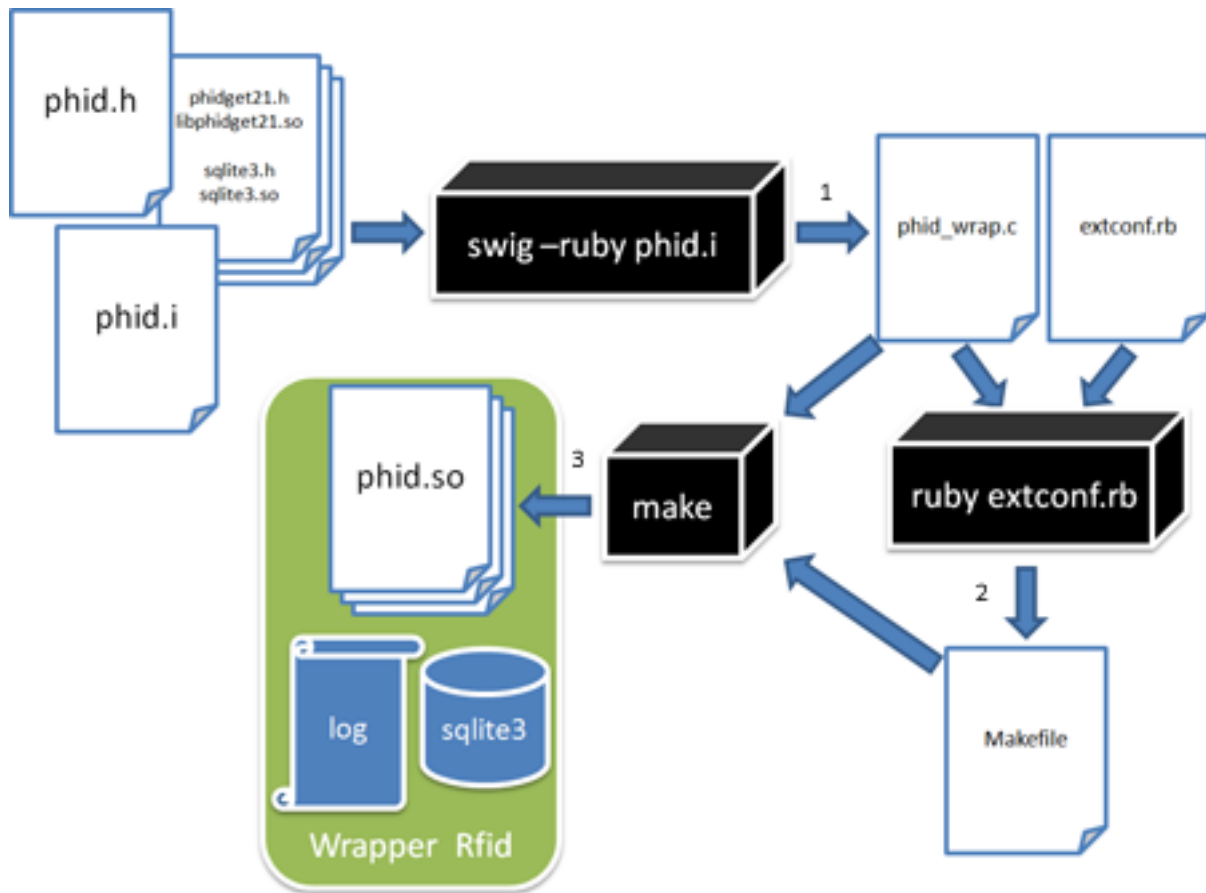


Figura 8.3: Insumos y procesos involucrados en la generación del wrapper

8.4.3 Codificación

Las funcionalidades descritas anteriormente se muestran a continuación:

```

1 // file: phid.h
2 ...
3 void enable_loggin(char *outputFile) {
4     CPhidget_enableLogging(level, outputFile);
5 }
6 void disable_loggin() {
7     CPhidget_disableLogging();
8 }
9 void log_message(char* id, char* message) {
10    CPhidget_log(level, id, message);
11 }
12 ...
13 int tag_handler(CPhidgetRFIDHandle RFID, void *usrptr, unsigned char *
    TagVal) {
14    ...
15    sprintf(id, "Reader_Serial:%d", get_serial_number());
  
```

```

16     sprintf(message, "Tag_Read:%02x%02x%02x%02x%02x", TagVal[0], TagVal[1],
    TagVal[2], TagVal[3], TagVal[4]);
17     sprintf(sql, "INSERT INTO log_VALUES('%d', '%02x%02x%02x%02x%02x',
    DATETIME('now'))", get_serial_number(), TagVal[0], TagVal[1],
    TagVal[2], TagVal[3], TagVal[4]);
18     printf("%s\n", message);
19     log_message(id, message);
20     rc = sqlite3_exec(db, sql, callback, 0, &zErrMsg);
21     ...
22 }
23 ...
24 void rfid_create(char *dbname) {
25     ...
26     rc = sqlite3_open(dbname, &db);
27     rc = sqlite3_exec(db, "CREATE TABLE IF NOT EXISTS log_(reader_id TEXT,
    NOT NULL, tag_read_id TEXT NOT NULL, read_at DATETIME NOT NULL,
    PRIMARY KEY_(reader_id, tag_read_id, read_at));", callback, 0, &
    zErrMsg);
28 }
29 ...

```

Siguiendo el proceso de generación del wrapper descrito anteriormente, en el que se utiliza el programa *SWIG* para, en este caso, generar un programa en C (**phid_wrap.c**), el cual, luego de ser compilado se genera la extensión de Ruby (**phid.so**).

Este programa en C generado por *SWIG*, define todos los métodos implementados en **phid.h** dentro de un módulo *Phid*:

```

1 // file:    phid_wrap.c
2 ...
3     mPhid = rb_define_module("Phid");
4     SWIG_InitializeModule(0);
5     for (i = 0; i < swig_module.size; i++) {
6         SWIG_define_class(swig_module.types[i]);
7     }
8     SWIG_RubyInitializeTrackings();
9     ...
10    rb_define_module_function(mPhid, "enable_login", _wrap_enable_login,
    -1);
11    rb_define_module_function(mPhid, "disable_login", _wrap_disable_login,
    -1);
12    rb_define_module_function(mPhid, "log_message", _wrap_log_message, -1);
13    ...
14    rb_define_module_function(mPhid, "tag_handler", _wrap_tag_handler, -1);
15    rb_define_module_function(mPhid, "tag_lost_handler",
    _wrap_tag_lost_handler, -1);
16    ...
17    rb_define_module_function(mPhid, "rfid_create", _wrap_rfid_create, -1);
18    ...

```

Este programa, además se adapta a las reglas que exige Ruby en cuanto a nombres, tipos y valores de variables, etc., y que puede ser compilado de acuerdo a otras reglas definidas en el archivo **Makefile**, generado por el módulo *mkmf*, con el cuál, finalmente, se genera la extensión de Ruby (**phid.so**) a través de la herramienta *make*.

8.4.4 Pruebas

Las pruebas de esta iteración se hicieron bajo el mismo ambiente descrito en la iteración 0.

Pruebas Funcionales

Para probar que el módulo *Phid* fue creado con todos sus métodos, se puede ejecutar en el *irb* el método **methods** del módulo:

```

1  \$ irb
2  irb(main):001:0> require 'phid'
3  => true
4  irb(main):002:0> Phid.methods.sort
5  => ["<", "<=", "<=>", "==", "===", "=~", ">", ">=", "__id__", "__send__",
    "ancestors", "autoload", "autoload?", "callback", "class", "class_eval", "class_variable_defined?", "class_variables", "clone", "const_defined?", "const_get", "const_missing", "const_set", "constants", "db", "db=", "disable_loggin", "display", "dup", "enable_loggin", "eql?", "equal?", "extend", "freeze", "frozen?", "get_device_name", "get_device_type", "get_device_version", "get_error_description", "get_serial_number" ...]
6  irb(main):003:0>

```

La siguiente prueba de funcionalidad consiste en verificar la inserción de los eventos de lectura en la base de datos. Para esto, primero se realizó el mismo procedimiento de pruebas llevado a cabo en la iteración 2 (Sección 8.3.4):

```

1  # file: ts_phidget_rfid_reader.rb
2  require 'phidget_rfid_reader'
3  p = PhidgetRfidReader.new
4  while true
5    cmd = gets
6    if cmd == 'close\n'
7      p.delete
8      break
9    end
10 end

```

La ejecución del programa anterior muestra el mismo resultado que se pudo apreciar en las pruebas de la iteración 1:

```

1  \$ ruby ts_phidget_rfid_reader.rb
2  Tag Read: 44a322e291
3  Tag Lost: 44a322e291
4  close

```

Luego, es posible comprobar lo siguiente:

- La base de datos y el archivo log fueron creados:

```
1 \ $ ls
2 phid.so    ts_phidget_rfid_reader.rb  phidget_rfid_reader.rb
      rfid_reader.rb  rfid_events.db  55841-phid.log
```

- La base de datos contiene el evento de lectura registrado:

```
1 # file: ts_query_sqlite.rb
2 require 'rubygems'
3 require 'sqlite'
4 db=SQLite3::Database.new('rfid_events.db')
5 cmd='select *_from_log'
6 db.execute(cmd) do |row|
7   row.each do |col|
8     print "#{col.to_s};"
9   end
10  print "\n"
11 end
```

El resultado de ejecutar el programa anterior muestra los registros que se generaron cuando el lector capturó la etiqueta *44a322e291*:

```
1 \ $ ruby ts_query_sqlite.rb
2 55841;44a322e291;2008-08-28 07:17:24;
```

- El contenido del archivo 55841-phid.log generado es el esperado:

```
1 ...
2 Thu Ago 28 07:17:22 2008,-1210390864,"clog.c(46)",INFO,"Enabling_
  logging"
3 Thu Ago 28 07:17:22 2008,-1210390864,"#<PhidgetRfidReader:0xb7c6f590>
  ",VERBOSE,"Device_Phidget_RFID_4-output::55841_Found!"
4 Thu Ago 28 07:17:24 2008,-1237988464,"Reader_Serial:55841",VERBOSE,"
  Tag_Read:44a322e291"
5 Thu Ago 28 07:17:24 2008,-1221203056,"Reader_Serial:55841",VERBOSE,"
  Tag_Lost:44a322e291"
6 Thu Ago 28 07:17:37 2008,-1210390864,"#<PhidgetRfidReader:0xb7c6f590>
  ",VERBOSE,"Device_Phidget_RFID_4-output::55841_Detached!"
7 ...
```

Como se puede apreciar, los resultados de las pruebas realizadas al wrapper al adaptador de lectura obtenidos fueron los esperados.

8.5 Iteración 4

Del 29-Ago-2008 al 05-Sep-2008

En esta iteración se desarrolló un servidor sencillo que permite hacer consultas y administrar los dispositivos lectores de manera remota.

8.5.1 Planificación

Para esta iteración se definió la meta “[Elaboración del manejador de eventos del middleware](#)”, con la cual se plantean los siguientes objetivos:

- Elaboración de un servidor que permita comandos de consulta y administración de manera remota
- Incorporar funciones en el servidor que permitan consultar eventos de lectura, administrar múltiples lectores, manejo de excepciones y manejo de concurrencia

8.5.2 Diseño

El principal problema que se plantea es la implementación de un servidor que permita hacer peticiones en forma de comandos sencillos para consultar eventos de lectura y administrar dispositivos.

Ruby permite implementar un servidor sencillo sobrescribiendo el método *run* de la clase *WEBrick::GenericServer*. *WEBrick* es un paquete que viene incluido en la librería estándar de Ruby, por lo que está disponible de manera automática al instalar cualquier distribución de Ruby.

El otro problema involucra la ejecución de consultas de eventos de lecturas. Para implementar esta funcionalidad se decidió permitir comandos en *SQL* ya que es un lenguaje estándar de consultas. De esta manera el cliente, al introducir un comando *SQL*, el servidor estará en la capacidad de devolver el resultado de tal consulta.

8.5.3 Codificación

Para implementar las funcionalidades requeridas en esta iteración, se definió la clase *RfidServer* de la siguiente manera:

```

1 # file : rfid_server.rb
2 ...
3 class RfidServer < WEBrick::GenericServer
4   def initialize (config={}, default=WEBrick::Config::General)
5     ...
6   end
7   def run(sock)
8     sock.print "#{Time.now.to_s}\n"
9     while true
10      cmd = sock.gets
11      case cmd
12      when /\bclose|exit|quit\b/
13        sock.print "Shutting_down..."
14        break
15      when /\battach\b/
16        ...
17      when /\b|list|readers\b/
```

```

18      ...
19      when /\ bdetach\s+\d+\W/
20      ...
21      when /\ bdetachall\W/
22      ...
23      when /\ bselect/
24      ...
25      when /\bh| help\W/
26      ...
27      else
28          sock.print "Unknown_command!\n"
29      end
30  end
31 end
32 end

```

El código anterior muestra los posibles comandos que admite la implementación del servidor:

close, exit o quit finalizan la conexión

attach instancia a un objeto de la clase *PhidgetRfidReader* y responde con un mensaje si se encontró el lector conectado

ls, list o readers muestra la lista de lectores que están conectados

detach # elimina el manejador del lector identificado con el serial #

detachall elimina todos los manejadores de los lectores conectados

select * hace una consulta a la base de datos (log) de eventos

help muestra un mensaje de ayuda explicando los comandos disponibles

Para iniciar el servidor se implementó el siguiente script:

```

1 # file: start.rb
2 require 'rfid_server'
3 s = RfidServer.new(:Port => 8004)
4 trap("INT"){ s.shutdown }
5 s.start

```

De esta manera, al ejecutar el programa, se inicia el servidor en el puerto indicado (en el ejemplo, el puerto 8004). A partir de este momento el servidor es capaz de recibir peticiones (comandos) y procesarlas de acuerdo a la lógica implementada en el método *run*

8.5.4 Pruebas

Para probar el correcto funcionamiento del servidor, simplemente se estableció una conexión a través de un cliente *telnet*, y se procedió a escribir comandos para verificar el comportamiento correcto del servidor, es decir, que las respuestas del servidor sean las esperadas.

Por ejemplo, se abrió una consola de comandos en la que se ejecutó el script para iniciar el servidor:

```
1 \ $ ruby start.rb
2 [2008-10-03 12:34:48] INFO WEBrick 1.3.1
3 [2008-10-03 12:34:48] INFO ruby 1.8.6 (2008-08-08) [i686-linux]
4 [2008-10-03 12:34:48] INFO RfidServer#start: pid=6394 port=8004
5 ...
```

En otra consola se utilizó el programa telnet y se escribieron comandos al servidor de la siguiente manera:

```
1 \ $ telnet localhost 8004
2 Trying 127.0.0.1...
3 Connected to 127.0.0.1.
4 Escape character is '^]'.
5 Thu Sep 04 18:51:15 -0430 2008
6 attach
7 Device 64526 found!
8 dettachment
9 Device 64526 detached!
10 ...
```

De esta manera se puede apreciar el correcto funcionamiento del servidor y los comandos implementados.

8.6 Iteración 5

Del 05-Sep-2008 al 12-Sep-2008

En esta iteración se incorporaron funcionalidades que permiten a un servidor comunicarse con otro servidor en caso de no poder procesar una petición de forma local. El mecanismo para identificar servidores adicionales interconectados sería a través un archivo de configuración general en el que se indiquen otro(s) servidor(es) como parte del sistema.

8.6.1 Planificación

Para esta iteración se definió la meta “Elaboración de la interfaz de nivel de aplicación del middleware”, con la cual se plantea el siguiente objetivo:

- Elaboración de un cliente telnet que permita la comunicación con otro server

8.6.2 Diseño

El principal desafío que se presenta con este objetivo es la implementación de un mecanismo estándar de comunicación que permita establecer un canal de comunicación con un servidor (*RfidServer*) que sea capaz de enviar peticiones en forma de comandos y recibir las respuestas esperadas. Para esto, se puede utilizar la librería *net/telnet* de Ruby.

Por otra parte, se plantea la necesidad de incluir una configuración que permita identificar cada uno de los servidores interconectados en el sistema RFID que se pueda estar implementado con el gem. Para implementar esta funcionalidad, se decidió implementar el archivo de configuración utilizando el formato YAML³, por su sencillez.

De esta manera se puede establecer un esquema de servidores interconectados como el siguiente:

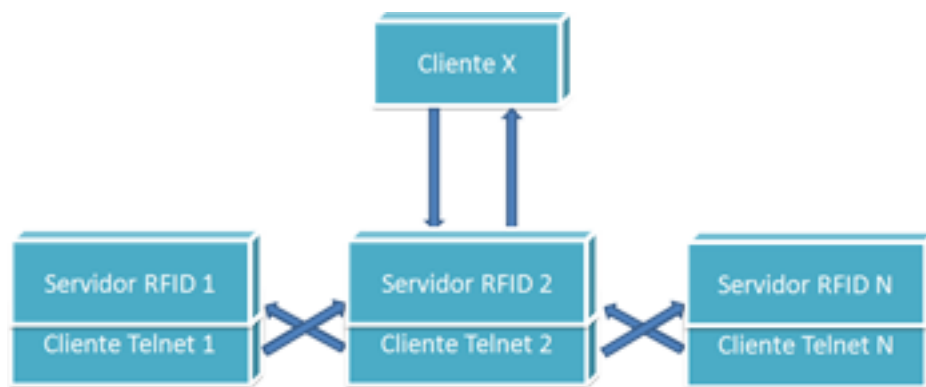


Figura 8.4: Esquema de conexión de servidores

Con este esquema se debe poder hacer una consulta a un servidor cualquiera, y este servidor, de no ser capaz de resolver localmente la consulta debe consultar a los demás servidores, los cuales podrán o no resolver la misma.

³Yet Another Markup Language—Otro Lenguaje de Marcado Más

8.6.3 Codificación

Para implementar el cliente telnet en Ruby es muy sencillo. Simplemente se agregaron los comandos necesarios utilizando la librería *net/telnet* de Ruby, la cual viene por defecto en todas las distribuciones de Ruby. De esta manera, la clase *RfidServer* quedó implementada de la siguiente manera:

```

1  # file: rfid_server.rb
2  ...
3  class RfidServer < WEBrick::GenericServer
4    def initialize(config={}, default=WEBrick::Config::General)
5      parse = YAML::parse(File.open("lib/config.yml"))
6      @local_config = parse.transform
7      p = @local_config["local_config"]["port"]
8      ip = @local_config["local_config"]["ip_address"]
9      config[:Port] = p unless p.nil?
10     config[:BindAddress] = ip unless ip.nil?
11     ...
12 end
13 def run(sock)
14   sock.print "#{Time.now.to_s}\n"
15   while true
16     cmd = sock.gets
17     case cmd
18     ...
19     when /\bselect/
20       begin
21         @db.execute(cmd) do |row|
22           row.each do |col|
23             sock.print "#{col.to_s};"
24           end
25           sock.print "\n"
26         end
27         @local_config.each do |key, value|
28           if key.include?("server")
29             #p value["ip_address"].to_s+" "+value["port"].to_s+"\n"
30             if !value["ip_address"].to_s.eql?(sock.peeraddr[3]) or !
31               value["port"].eql?(sock.peeraddr[1])
32               conection = Net::Telnet::new("Host" => value["ip_address"]
33                 .to_s, "Timeout" => 10, "Port" => value["port"])
34               sock.print "Server_"+value["ip_address"].to_s+": "+value["
35                 port"].to_s+"_"
36               conection.cmd("consult_#{cmd.chop}") do |response|
37                 sock.print response.to_s
38               end
39               conection.close
40             end
41           end
42         end
43       end
44     rescue SQLite3::SQLException => e

```

```

41         sock.print "#{e}\n"
42     rescue Exception => e
43         sock.print "#{e}\n"
44     end
45     ...
46 end
47 end
48 end
49 end

```

En el método *initialize* se puede apreciar que se implementó una lógica especial para obtener la configuración de los distintos servidores del sistema. Esta configuración se implementó para que se definiera a través de un archivo `config.yml`, usando una estructura como la siguiente:

```

1 # file: config.yml
2 local_config:
3   id_server: 0
4   ip_address: 127.0.0.1
5   port: 8004
6 server1:
7   id_server: 1
8   ip_address: 127.0.0.1
9   port: 8000

```

En este ejemplo se está indicando que se configurarán dos servidores en la máquina local. Uno escuchando por el puerto 8000 y otro por el puerto 8004. De esta manera, si se desean agregar más servidores, simplemente basta con agregar una configuración adicional, indicando el *id_server*, *ip_address* y *port*.

Es necesario especificar al menos una configuración para el servidor *local_config* y para cada servidor adicional se debe indicar con la etiqueta *server#*, donde *#* puede ser un número que identifique unívocamente a un servidor dentro del sistema.

Finalmente, en el método *run*, en el comando *select* se implementó la lógica que establece la conexión con los demás servidores especificados en el archivo `config.yml`.

8.6.4 Pruebas

Para comprobar el comportamiento adecuado de la funcionalidad implementada se instanciaron dos servidores utilizando el script `start.rb`:

En una consola se ejecutó el siguiente script:

```

1 # file: start1.rb
2 require 'rfid_server'
3 s = RfidServer.new(:Port => 8004)
4 trap("INT"){ s.shutdown }
5 s.start

```

En otra consola se ejecutó este otro script:

```

1 # file: start2.rb
2 require 'rfid_server'
3 s = RfidServer.new(:Port => 8000)

```

```
4 trap("INT"){ s.shutdown }
5 s.start
```

Es importante ejecutar los scripts en directorios diferentes ya que, de lo contrario, la base de datos que se genera sería la misma, por que se estuviese consultando doblemente la misma información.

Una vez iniciados los servidores, simplemente se puede establecer una conexión a cualquiera de los servidores y hacer una petición sencilla al servidor. Por ejemplo:

```
1 \$ telnet localhost 8004
2 Trying 127.0.0.1...
3 Connected to 127.0.0.1.
4 Escape character is '^]'.
5 Thu Sep 04 18:51:15 -0430 2008
6 select * from log;
7 64526;17002f3f02;2008-11-09 17:19:40
8 64526;17002f3f02;2008-11-09 17:20:23
9 ...
```

Sabemos que el server que está escuchando por el puerto 8004 no resolverá la consulta porque su base de datos está vacía.

8.7 Iteración 6

Del 12-Sep-2008 al 26-Sep-2008

En esta iteración se procedió a empaquetar el código bajo la estructura de un gem de Ruby.

8.7.1 Planificación

Para esta iteración se definió la meta “[Crear el gem final con la extension y los modulos del middleware](#)”, con la cual se plantearon los siguientes objetivos:

- Crear el gemspec
- Definir la estructura general de gem
- Crear los test para el gem
- Empaquetar (construir) el gem
- Importar el proyecto en RubyForge

8.7.2 Diseño

RubyGems establece que todo gem debe tener la siguiente estructura en cuando a sistema de archivos:

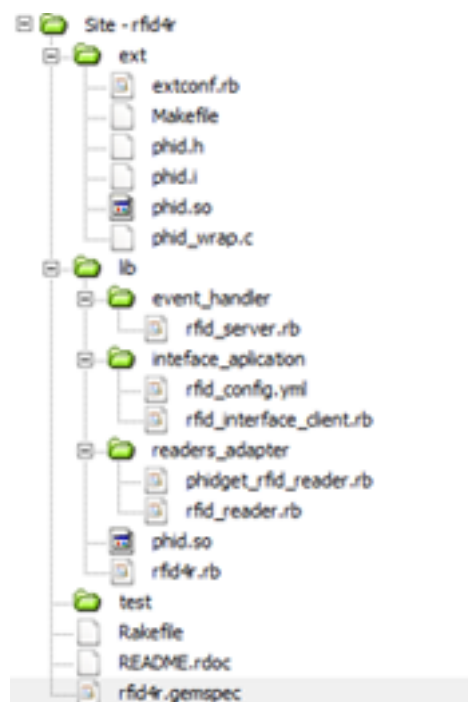


Figura 8.5: Árbol de archivos del gem

Esta estructura sigue las convenciones descritas en la Sección 5.2.3. En base a esta estructura, se definen las reglas necesarias para definir la especificación del gem en el archivo *gemspec*.

8.7.3 Codificación

En esta iteración, la única actividad de codificación necesaria fue la elaboración de la especificación del gem en el archivo `rfid4r.gemspec`:

```

1 # file: rfid4r.gemspec
2 require 'rubygems'
3 spec = Gem::Specification.new do |s|
4   s.name = "rfid4r"
5   s.version = "1.0.0"
6   s.author = "Daniel_González"
7   s.email = "dangt85@gmail.com"
8   s.homepage = "http://rfid4r.rubyforge.org"
9   s.platform = Gem::Platform::CURRENT
10  s.summary = "An_RFID_middleware_using_a_Ruby_wrapper_for_the_
    PhidgetRFID-1023_reader."
11  s.files = Dir.glob("{doc,ext,lib,test}/**/*")
12  s.add_dependency("sqlite3-ruby", ">=1.2.4")
13  s.has_rdoc = true
14  s.extra_rdoc_files = ['README.rdoc']
15 end

```

Una vez especificadas las reglas necesarias para indicar a RubyGems cómo empaquetar el código se puede ejecutar el siguiente comando:

```

1 \$ gem build rfid4r.gemspec
2 Successfully built RubyGem
3 Name: rfid4r
4 Version: 1.0.0
5 File: rfid4r-1.0.0-x86-linux.gem

```

Luego, es posible instalar el gem con el siguiente comando:

```

1 \$ gem install rfid4r-1.0.0-x86-linux.gem
2 Successfully installed rfid4r-1.0.0-x86-linux
3 1 gem installed
4 Installing ri documentation for rfid4r-1.0.0-x86-linux...
5 Installing RDoc documentation for rfid4r-1.0.0-x86-linux...

```

8.7.4 Pruebas

Las pruebas en esta iteración son bastante sencillas. Basta con ejecutar en una consola, utilizando el programa *irb*:

```

1 \$ irb
2 irb (main):001:0> require 'rubygems'
3 => true
4 irb (main):002:0> require 'rfid4r'
5 => true
6 [2008-09-25 16:14:02] INFO WEBrick 1.3.1
7 [2008-09-25 16:14:02] INFO ruby 1.8.6 (2008-08-08) [i686-linux]
8 [2008-09-25 16:14:02] INFO RfidServer#start: pid=7198 port=8000

```

9 . . .

De esta manera se puede constatar simplemente que funciona el paquete instalado bajo la estructura de un gem de Ruby.

La instalación es muy sencilla, simplemente se debe descargar el gem con los archivos necesarios e instalarlo⁴ usando los comandos provistos por RubyGems. Por ejemplo, una vez descargado el paquete `rfid4r-1.0.0-x86-linux.gem` basta con ejecutar el siguiente comando.

```
1 \ $ gem install rfid4r-1.0.0-x86-linux.gem
```

Para el correcto funcionamiento de este gem, será necesario instalar en el ambiente de desarrollo, la librería del lector PhidgetRFID-1023⁵ y la librería SQLite3⁶ para manejo de bases de datos. Por otra parte, este gem depende del gem `sqlite3-ruby`⁷ para un funcionamiento óptimo en el manejo de la base de datos SQLite.

⁴disponible para descargar desde <http://rfid4r.rubyforge.org/>

⁵disponible para descargar desde <http://www.phidgets.com/>

⁶disponible para descargar desde <http://www.sqlite.org/>

⁷disponible para descargar desde <http://rubyforge.org/projects/sqlite-ruby/>

8.8 Iteración 7

Del 26-Sep-2008 al 03-Oct-2008

Esta iteración se dedicó a hacer refactorizaciones del código para hacer mejoras generales del sistema.

8.8.1 Planificación

Para esta iteración se definió la meta “[Modificaciones y pruebas del gem](#)”, con la cual se definieron los siguientes objetivos:

- Separar el servidor webrick del cliente telnet
- Modificar métodos para optimizar algoritmos o mejorar mensajes, comandos, opciones, etc.

8.8.2 Diseño

El problema principal que se plantea con la modificación planteada en los objetivos definidos es la especificación de la estructura final del gem, en base a la cual se desea explicar detalladamente cómo esta constituido cada componente del middleware RFID:

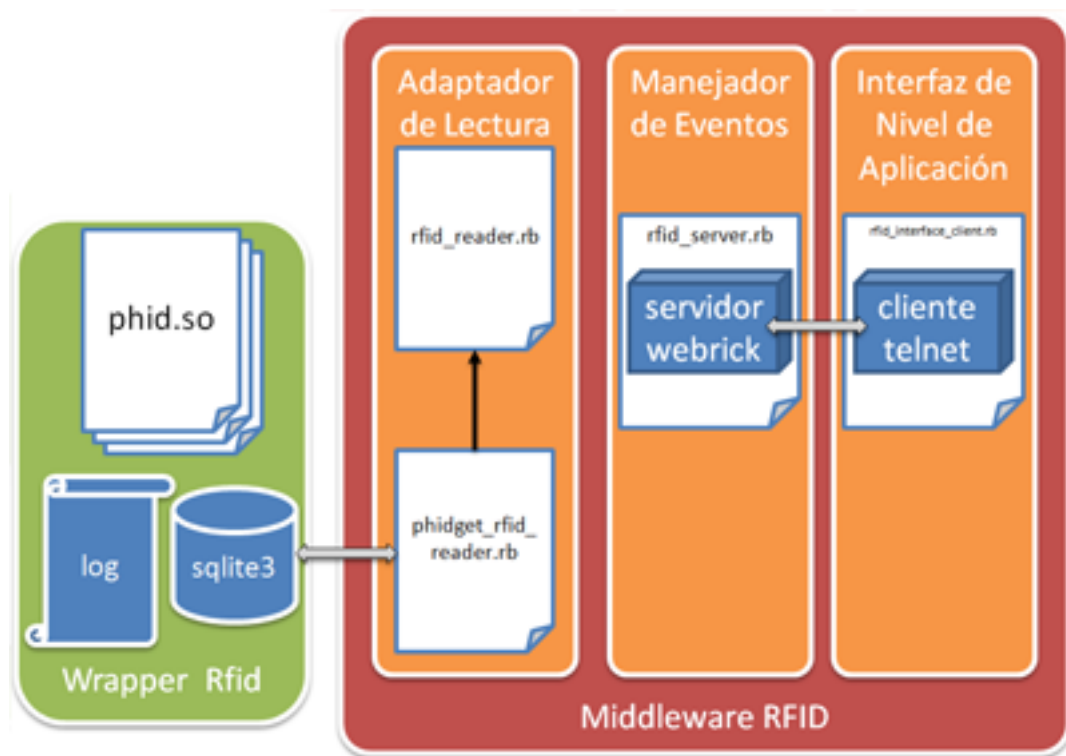


Figura 8.6: Estructura final del gem

Esta estructura muestra de manera sencilla en qué componente se pueden ubicar los módulos y clases de Ruby que se tienen hasta ahora.

8.8.3 Codificación

La principal modificación consiste en separar el servidor webrick del cliente telnet implementado. Otras modificaciones consisten en agregar los comentarios necesarios para cumplir con los requerimientos de RubyGems para generar la documentación RDoc del gem:

```

1  # file: rfid_server.rb
2  ...
3  #
4  # La clase RfidServer representa el manejador de eventos del Middleware
   RFID, para el manejo de los
5  # eventos generados por las capturas de etiquetas RFID.
6  # RfidServer es una clase concreta de la clase WEBrick::GenericServer que
   implementa un servidor
7  # para el manejo de un lector RFID. Adicionalmente recibe comandos que
   permiten consultar,
8  # registrar y liberar el lector RFID, asi como tambien comandos para
   consultar la base de datos SQLite
9  # de eventos RFID.
10 #
11 class RfidServer < WEBrick::GenericServer
12   #Representa una referencia a una interfaz cliente para consultas hacia
   otros servidores RFID
13   attr_reader :interface_client
14   #Referencia al lector RFID conectado al servidor
15   attr_reader :reader
16   #Referencia logica para la presencia o no del lector RFID
17   attr_reader :is_attach
18   #
19   # Crea un nuevo servidor RFID utilizando el archivo de configuracion "
   rfid_config.yml", adicionalmente
20   # instancia la base de datos SQLite de eventos para su uso porterior.
21   #
22   def initialize(config={}, default=WEBrick::Config::General)
23     @interface_client = RfidInterfaceClient.new
24     p = @interface_client.config["local_config"]["port"]
25     ip = @interface_client.config["local_config"]["ip_address"]
26     config[:Port] = p unless p.nil?
27     config[:BindAddress] = ip unless ip.nil?
28     @is_attach = false;
29     ...
30   end
31   #
32   def update(event)
33     ...
34     when /\bselect/
35       begin
36         @db.execute(cmd) do |row|
37           row.each do |col|

```

```

38         sock.print "#{col.to_s};"
39     end
40     sock.print "\n"
41 end
42 @interface_client.consult(sock,cmd)
43 rescue Exception => e
44     sock.print "#{e}\n"
45 end
46 ...
47 end
48 #
49 # Recibe y procesa los comandos, a continuacion se muestra una lista de
    los comandos soportados
50 # y una breve descripcion de su funcion:
51 #
52 #   attach:           Attach a new RFID reader
53 #   detach [serial_id]: Detach a RFID reader
54 #   detachall:        Detach all RFID readers
55 #   readers:          List all RFID readers
56 #   select [SQL]:      Consulta de tipo SQL a la base de datos SQLite
57 #   quit/close/exit:   Close the connection with de server
58 #
59 def run(sock)
60     ...
61 end
62 end

```

En la clase anterior se puede apreciar que se agregaron los comentarios pertinentes para la documentación de la clase y las otras modificaciones que consisten en la separación del servidor del cliente se pueden apreciar en las porciones presentes en el método `initialize` y la implementación del comando `select`.

```

1 # file: rfid_interface_client.rb
2 ...
3 #
4 # La clase RfidInterfaceClient representa un cliente interno en el
    servidor, el cual realiza peticiones a otros servidores
5 #
6 class RfidInterfaceClient
7     # Contiene los parámetros de configuración obtenidos del archivo
        rfid_config.yml
8     attr_reader :config
9     #
10    # Crea un nuevo cliente a partir del archivo rfid_config.yml
11    #
12    def initialize(yml='rfid_config.yml')
13        parse = YAML::parse(File.open(yml))
14        @config = parse.transform
15    end

```

```

16  #
17  # ejecuta el comando cmd y lo transmite al socket sock
18  #
19  def consult (sock,cmd)
20      ...
21  end
22 end

```

La clase anterior permite instanciar un cliente telnet el cuál realizará peticiones a los servidores configurados en el archivo `rfid_config.yml` a través del comando `consult`.

8.8.4 Pruebas

Para hacer las pruebas finales y comprobar las características que introduce el concepto cliente-servidor, se pueden instanciar dos servidores. Para esto es necesario configurar dos servidores.

En una consola se ejecuta el siguiente comando para iniciar un servidor configurado para ejecutarse en el puerto 8000:

```

1  \rfid4r1\$ ruby rfid4r.rb
2  [2008-09-25 16:14:02] INFO  WEBrick 1.3.1
3  [2008-09-25 16:14:02] INFO  ruby 1.8.6 (2008-08-08) [i686-linux]
4  [2008-09-25 16:14:02] INFO  RfidServer#start: pid=7198 port=8000
5  ...

```

En otra consola, se ejecuta el mismo comando, pero esta vez en un servidor configurado para iniciarse en un puerto distinto (8804):

```

1  \rfid4r2\$ ruby rfid4r.rb
2  [2008-09-25 16:14:45] INFO  WEBrick 1.3.1
3  [2008-09-25 16:14:45] INFO  ruby 1.8.6 (2008-08-08) [i686-linux]
4  [2008-09-25 16:14:45] INFO  RfidServer#start: pid=7691 port=8004
5  ...

```

Luego, se procede a conectarse al servidor a través de un cliente telnet:

```

1  \$ telnet localhost 8000
2  Trying 127.0.0.1...
3  Connected to 127.0.0.1.
4  Escape character is '^]'.
5  Thu Sep 04 18:51:15 -0430 2008
6  attach
7  Device 55841 found!
8  select * from log;
9  64526;44a322e291;2008-11-09 17:20:40
10 64526;44a322e291;2008-11-09 17:37:24

```

Con otro cliente telnet se puede establecer una conexión al segundo servidor configurado:

```

1  \$ telnet localhost 8004
2  Trying 127.0.0.1...
3  Connected to 127.0.0.1.
4  Escape character is '^]'.
5  Thu Sep 04 18:51:15 -0430 2008

```

```
6 attach
7 Device 64526 found!
8 select * from log;
9 55841;17002f3f02;2008-11-09 17:19:40
10 55841;17002f3f02;2008-11-09 17:20:23
11 Server 127.0.0.1:8000
12 64526;44a322e291;2008-11-09 17:20:40
13 64526;44a322e291;2008-11-09 17:37:24
```

En este caso se puede apreciar en el segundo cliente telnet, el cual está conectado al servidor que está escuchando por el puerto 8004, que se ha hecho una consulta al servidor que está escuchando por el puerto 8000 (*Server 127.0.0.1:8000*).

Es importante destacar que un servidor sólo permite manejar un lector. Por lo tanto es necesario iniciar tantos servidores como lectores se pretendan conectar a la máquina.

Por otra parte, un aspecto que se puede resaltar del código anterior (**rfid_server.rb**) es que se agregaron los comentarios requeridos por RDoc para generar la documentación necesaria al momento de generar el gem.

Parte IV

Conclusiones

Capítulo 9

Conclusiones

Este trabajo se enmarcó en la modalidad de investigación y desarrollo, con la finalidad de disponer de las soluciones y herramientas que permitieran cumplir con los objetivos planteados en la propuesta. El resultado fue un producto de software llamado *rfid4r*, destinado a proveer un middleware para sistemas RFID.

Este producto se desarrolló utilizando el lenguaje Ruby y el lenguaje C para la elaboración de una extensión de Ruby que permitiera el manejo del dispositivo PhidgetRFID-1023. El producto fue empaquetado utilizando la propuesta de Ruby para desarrollo, distribución de productos y librerías de software. Esta propuesta es la de RubyGems, que no es más que un sistema para administrar paquetes de software desarrollados en Ruby.

La intención de desarrollar este producto de software es proveer un mecanismo práctico para implementar sistemas RFID. El enfoque se hizo sobre un componente muy importante en este tipo de sistemas: el middleware RFID. Este componente provee una capa de abstracción que permite a las demás aplicaciones RFID poder manipular los dispositivos RFID.

De esta manera, se facilita considerablemente la implementación de sistemas que pueden ser utilizados para diversos fines y actividades. Como se explicó en diversas ocasiones, la tecnología RFID puede ser utilizada para identificar objetos, personas, animales, etc. Pero esta sencilla actividad de identificación se puede convertir en un desafío cuando están involucrados grandes cantidades de objetos identificables. RFID ha probado ser una tecnología novedosa e innovadora en este tipo de actividades.

Por estos motivos se decidió implementar el componente middleware utilizando el proceso de desarrollo XP y la metodología AM. El proceso de desarrollo XP simplificó las actividades de planificación, codificación, y pruebas, con lo cual se redujo el tiempo de implementación. Lo cual trajo como consecuencia que se pudiese dedicar más tiempo en actividades de pruebas, modificaciones y diseño. Esto a su vez, redundó en una mejoría en la calidad del producto, con lo cual se pudo lograr producir un software robusto, efectivo y eficiente.

Además, gracias a la naturaleza del lenguaje Ruby, el producto es fácilmente extensible y adaptable, ya que contiene pocas líneas de código. Esto se puede percibir como una ventaja porque resulta en un código simple de entender, fácil de modificar y extender. También, al tener menos líneas de código, existen menos posibilidades de errores y, a la vez, es más fácil conseguirlos.

Por otra parte, la metodología AM también simplificó el desarrollo, ya que la actividad de diseño no representó un obstáculo considerable. Ya que con otras metodologías, se requieren modelos, diagramas, maquetas, etc., que siguen un estándar específico y que pocas veces se adapta fácilmente a la solución

que se está buscando. Con la metodología AM, la actividad de diseño se simplifica en cuanto a la documentación, diagramación y modelado. De esta manera se puede ahorrar tiempo y aprovecharlo en mejorar el producto en sí, hacer más propuestas, extender las funcionalidades y hacer más pruebas.

Las dificultades que se presentaron en el desarrollo fueron fácilmente superadas gracias a la implementación de las actividades propuestas por XP y AM, y gracias a la utilización de las tecnologías propuestas que permitieron una fácil integración y un desarrollo de un producto adaptable, mantenible y extensible.

En cuanto al gem, es necesario destacar que está implementado para utilizar la arquitectura cliente-servidor. En cuanto al servidor, es importante destacar que el protocolo implementado está basado en comandos simples y las respuestas son enviadas en formato de texto plano. El protocolo de comunicación es transparente al programador ya que tales funcionalidades están implementadas por la clase *Webrick::GenericServer*. Las conexiones se establecen a través de sockets TCP. Por lo tanto, se puede decir que el protocolo utilizado es simplemente un programa basado en comandos sencillos.

En vista de la naturaleza distribuida que introduce el concepto cliente-servidor, las pruebas realizadas tuvieron que elaborarse de manera convencional por razones de tiempo y simplicidad. Es decir, que no se utilizaron los frameworks de pruebas disponibles para Ruby.

Un aspecto que hay que resaltar es la posibilidad de extender las funcionalidades del gem, en especial, la posibilidad de dar soporte para el manejo de otros dispositivos lectores. Para esto, será necesario generar una extensión para Ruby, siguiendo, en la medida de lo posible, algunas de las convenciones utilizadas para el caso del dispositivo PhidgetRFID.

La posibilidad de extender el soporte para nuevos lectores podrá permitir añadir funcionalidades más complejas como las que involucran el uso de lectores para etiquetas activas. De esta manera, se contemplaría la posibilidad de que los lectores pueden leer etiquetas RFID y a su vez, escribir o sobrescribir datos en ellas.

También será posible administrar las etiquetas activas de última generación desde el middleware, de manera que las mismas actúen como lectores de otras etiquetas (Etiquetas EPC y de Clase IV y V).

Es importante destacar que el desarrollo del producto estuvo enfocado hacia la plataforma Linux. Sin embargo, el proceso de adaptación del paquete a otras plataformas como Windows y Mac es muy similar. Para el caso de Windows es necesario generar una librería compartida (por ejemplo, *phid.dll*) y para el caso de Mac se requiere un objeto *bundle* (por ejemplo, *phid.bundle*). Adicionalmente, para que el gem pueda ser instalado correctamente es necesario configurar en el repositorio¹ las opciones para el tipo de plataforma. De esta manera, un usuario que intente instalar el gem podrá seleccionar la plataforma que desee al momento de descargar el paquete.

Finalmente, se puede concluir que los objetivos planteados para el desarrollo de este trabajo fueron alcanzados, porque se logró desarrollar un gem de Ruby con el cual es posible implementar un middleware RFID basado en la arquitectura cliente-servidor. Específicamente, este gem puede manejar los dispositivos lectores de la familia PhidgetsRFID-1023. Por otra parte, se cumplió con el objetivo de publicar el gem bajo una licencia de código abierto, permitiendo a otros programadores continuar el desarrollo y extender las funcionalidades del producto.

9.1 Mejoras a Futuro

Es necesario resaltar que el producto desarrollado puede ser extendido si se consideran algunas ideas que permitan agregar soporte para nuevas funcionalidades y características en el middleware RFID que implementa el gem. Algunas ideas pueden ser:

¹<http://rfid4r.rubyforge.org>

- Implementar el gem contemplando el soporte para las plataformas Windows y Mac
- Agregar soporte para más dispositivos lectores-escritores, etiquetas activas y futuras generaciones de dispositivos y etiquetas
- Agregar soporte para servicios web
- Implementar módulos usando técnicas avanzadas de programación como Comet²
- Mejores interfaces de comunicación implementando nuevos protocolos, comandos y agregando políticas y niveles de seguridad

9.2 Recomendaciones

Es importante destacar algunas consideraciones y recomendaciones para lograr el mejor provecho del gem:

- Revisar la documentación *ri* y *rdoc* para entender el funcionamiento del gem
- Editar el archivo `rfid_config.yml` para configurar tantos servidores como sean necesarios
- Es necesario instanciar un servidor RFID por cada lector que se quiera utilizar
- Mantener al día, desarrollar pruebas constantemente, actualizar y mejorar el gem desde su repositorio <http://rfid4r.rubyforge.org>

Otras recomendaciones referentes a las aplicaciones externas que utilicen el servidor RFID pueden contemplar lo siguiente:

- Realizar operaciones con bases de datos o repositorios LDAP para relacionar o asociar datos e información
- Generación de archivos XML y reportes en PDF
- Integración con APIs de GoogleMaps para implementar sistemas de posicionamiento en tiempo real
- Generación de gráficas y estadísticas

9.3 Aportes de la Investigación

La investigación realizada permitió entender y exponer ciertos aspectos involucrados en la elaboración de un middleware RFID en Ruby. Algunos de estos aspectos son:

- La implementación y adaptación de metodologías y procesos de desarrollo como XP y AM para la implementación de un producto de software
- El desarrollo de una extensión de Ruby utilizando SWIG y el módulo *mkmf*
- El desarrollo de un programa Cliente-Servidor sencillo en Ruby

²también conocido como server push, HTTP push, HTTP streaming, Pushlets, Reverse Ajax, y otros

- El desarrollo de un middleware RFID para administrar dispositivos y hacer consultas
- El desarrollo de un gem de Ruby

Apéndice

Apéndice A

Dispositivos RFID

A.1 PhidgetRFID — 1023

En esta sección se describirán a manera de demostración, las características técnicas del dispositivo lector PhidgetRFID — 1023, así como también unas instrucciones breves de instalación y algunos conceptos de programación del dispositivo. Esta misma información e información adicional se puede encontrar en <http://www.phidgets.com>.

A.1.1 Características del Producto

- Lee etiquetas dentro de un rango de 3 pulgadas de distancia
- Lee etiquetas usando el protocolo EM4102
- Retorna el identificador único contenido en la etiqueta
- Provee 2 salidas digitales para LEDs¹, relays, etc.
- LED incorporado en la tarjeta
- Conexión directa a computadoras usando un puerto USB

Para que un lector RFID como el PhidgetRFID pueda comunicarse con una etiqueta RFID, deben compartir un mismo protocolo de comunicación. Este protocolo actúa como una serie de reglas para definir la manera en cómo se transmite la data de manera inalámbrica entre el lector y la etiqueta. El PhidgetRFID, así como todas las etiquetas vendidas por Phidgets, usan el protocolo EM4102. Cualquier otra etiqueta que use el protocolo EM4102 puede ser usada con el lector PhidgetRFID. Todas las etiquetas RFID vendidas por Phidgets tienen la garantía de ser únicas y están disponibles las siguientes presentaciones:

- Etiquetas de disco de 30mm (estas etiquetas se pueden encontrar en prendas de vestir, incorporadas en objetos). Ver Figura A.1



Figura A.1: Etiquetas de disco de 30mm. Tomado de <http://www.phidgets.com>



Figura A.2: Etiquetas en forma de tarjetas de crédito. Tomado de <http://www.phidgets.com>

- Etiquetas en forma de tarjetas de crédito (buenas para aplicaciones de identificación de seguridad). Ver Figura A.2
- Etiquetas en forma de llaveros (incorporadas fácilmente en llaves)

Como se describió en el Capítulo 4, las etiquetas activas usan su propia fuente de poder para activar su antena y poder comunicarse y transmitir datos. Las etiquetas pasivas convierten la señal emitida por el lector RFID para activar su antena, por lo que éstas etiquetas son más baratas y más fáciles de implementar. Sin embargo, las etiquetas pasivas requieren una señal de radiofrecuencia más fuerte para operar y su rango efectivo se limita a un área muy cercana. En el caso del PhidgetRFID, las etiquetas pueden ser leídas dentro de un rango de 3 pulgadas de proximidad. La forma, el tamaño y el material de la etiqueta también afecta la distancia de efectividad de lectura, así como la orientación que se le debe dar al lector y la etiqueta en el ambiente operativo.

El uso de múltiples lectores PhidgetRFID dentro de un área de 1 a 2 metros, puede ocasionar interferencia. Esto se puede superar a nivel de software, habilitando el secuenciamiento de antenas de lectores individuales. Esto es:

- empezar con todos los lectores deshabilitados
- habilitar la antena del primer lector
- esperar 100 milisegundos o más para detectar cualquier etiqueta
- deshabilitar la antena del primer lector y habilitar la segunda
- y repetir el ciclo de espera

El lector PhidgetRFID no ofrece mecanismos para eludir o para detectar de colisiones. Si dos etiquetas se encuentran dentro del área de lectura del lector al mismo tiempo, ninguna de las etiquetas serán detectadas. Una etiqueta RFID debe ser removida del área de lectura para que la segunda pueda ser introducida.

¹Light Emitting Diode — Diodo Emisor de Luz

El lector PhidgetRFID tiene cuatro salidas, dos de las cuales están disponibles para el usuario y las otras dos son para el control interno de la tarjeta Phidget. La salida 0 es una fuente de +5V desde el bus USB a través de un P-Channel MOSFET con menos de 1 ohm de impedancia. Esto puede ser usado para intercambiar un dispositivo TTL o CMOS, o para dirigir un relay 5VDC como el Aromat JS1-5v. La salida 1 es un LED de 5VDC con un máximo de 15mA de corriente (salida de 250 ohm CMOS). Ambas salidas 0 y 1 están disponibles en el hardware en los bloques terminales de la tarjeta PhidgetRFID. Si la salida 0 es usada para dirigir un relay, se debe colocar un diodo en el relay como se muestra en la Figura A.3.

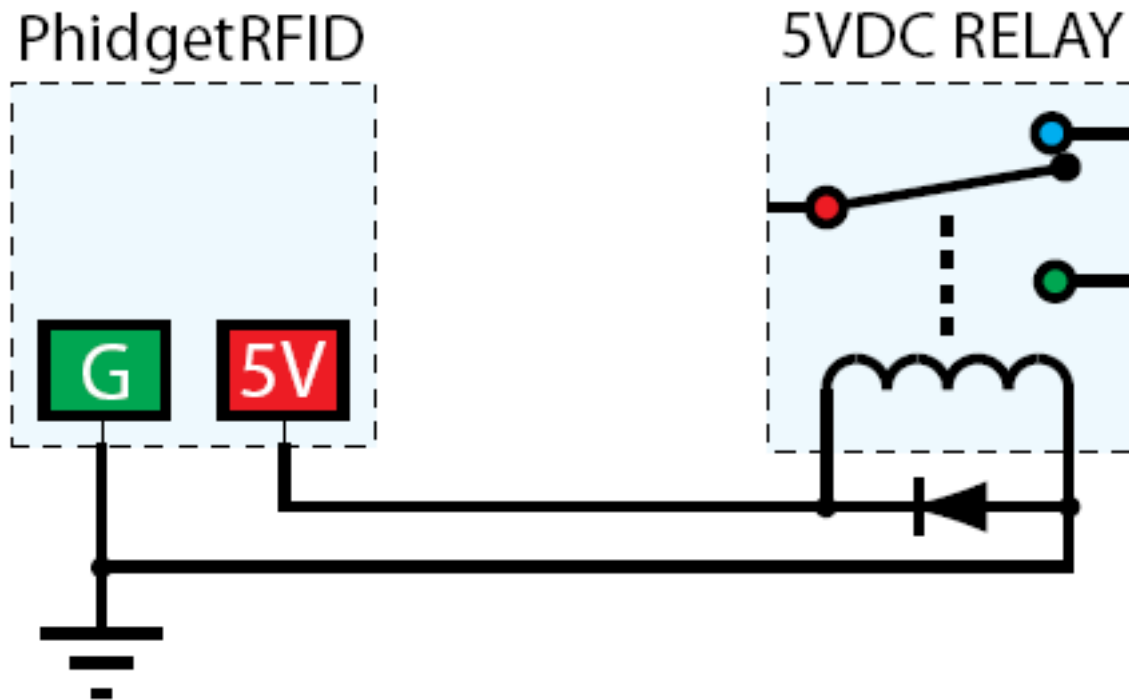


Figura A.3: Puente con diodo en relay. Tomado de <http://www.phidgets.com>

Si esto no se hace se pueden ocasionar daños permnentes en la tarjeta PhidgetRFID. El lector PhidgetRFID viene equipado con un LED integrado a la tarjeta que puede ser controlado usando la propiedad LED (a nivel de software). Adicionalmente, la antena integrada a la tarjeta puede ser habilitada y deshabilitada usando la propiedad RF Enable (también a nivel de software). La antena debe ser habilitada por el lector PhidgetRFID para detectar y leer etiquetas RFID.

A.1.2 Ambiente de Programación

La filosofía que sigue Phidget es que no se ser un ingeniero eléctrico para hacer proyectos que usen dispositivos como sensores, motores, controladores de motores e interfaces de tarjetas. Lo que se necesita saber es cómo programar. Phidget tiene varias APIs disponibles con soporte para los sistemas operativos: Windows 2000/XP/Vista, Windows CE, Linux, y Mac OS X. En cuanto a los lenguajes de programación y frameworks, las APIs de Phidget tienen soporte para: VB6, VB.NET, C#.NET, C++, Flash 9, Flex, Java, LabVIEW, Python, Max/MSP y Cocoa.

Phidget ha diseñado las librerías para proveer la mayor libertad para que el programador pueda usar el modelo que desee. Para esto, se implementaron las librerías como una serie de capas alrededor

del API de C. La API de C puede ser programada para usarse en Windows, CE, OS X y Linux. Con la API de C, C/C++, es posible escribir código con soporte multiplataforma. Para sistemas con recursos limitados, la API de C puede ser la única opción. La API de Java está construida sobre la API de C. Java es multiplataforma, pero es posible que no pueda ser soportada por plataformas particulares como CE. La API de .NET está dirigida para el framework .NET 2.0, pero también tiene soporte para .NET 1.1 y .NET Compact Framework. La API COM está programada para codificar en VB6, VBScript, Excel (VBA), Delphi y Labview. La librería ActioScript 3.0 usa el enlace de comunicación PhidgetWebService para ser usada en Flex y Flash 9.

A.1.3 Recomendaciones para la Programación

- Cada Phidget tiene un serial único, lo que permite ordenar por dispositivo para saber cuál es cuál en tiempo de ejecución. A diferencia de los dispositivos USB que se modelan así mismos como puertos COM, no es necesario preocuparse por saber dónde, en el bus USB, se conecta el Phidget. Si se tienen más de un Phidget, incluso del mismo tipo, sus seriales permiten ser ordenados en tiempo de ejecución.
- Cada Phidget conectado es controlado desde la aplicación, usando un objeto específico para ese Phidget. Este enlace entre el Phidget y el objeto de software es creado cuando se hace la llamada al grupo de comandos `.OPEN`. Esta asociación se mantendrá, incluso si el Phidget es desconectado y reconectado nuevamente, hasta que se llame a `.CLOSE`.

Existe una aplicación hecha por Phidgets Inc. llamada PhidgetWebService, que actúa como un proxy de red en una computadora. Esta aplicación permite a otras computadoras conectadas en una red, comunicarse con Phidgets conectados en otras computadoras. Todas las APIs permiten comunicarse con Phidgets en otras computadoras que tengan ejecutando el PhidgetWebService. Esta aplicación también permite la comunicación con otras aplicaciones que también estén conectadas al PhidgetWebService, a través del objeto PhidgetDictionary.

Para mayor información acerca de las APIs, métodos de instalación, instrucciones detalladas, etc., existe un manual en Inglés que se puede descargar en formato .pdf desde la dirección <http://www.phidgets.com/documentation/Phidgets/1023.pdf>.