Raúl Broseta Gutiérrez

DISPOSITIVOS MÓVILES Y NFC APLICADOS AL CANJEO DE TICKETS: BeepVip

TESIS DE MÁSTER

dirigida por el Dr. Joan Fons i Cors

Departamento de Sistemas
Informáticos y Computación

Valencia, Septiembre 2012

Agradecimientos

Especialmente a Belén por su apoyo en los buenos y los malos momentos, confianza absoluta y especial cariño.

A mi familia, sobre todo a mis padres y mis hermanos, por su apoyo, cariño y su paciencia.

A Joan Fons por su comprensión, dedicación y asesoramiento.

A mis compañeros y profesores del máster en Ingeniería del Software, Métodos Formales y Sistemas de Información por hacer de él una experiencia inolvidable.

Indice

Tabla de figuras

1 Introducción

El presente trabajo se enmarca dentro de la tesis de master necesaria para conseguir el título del Master en Ingeniería de Software, Métodos Formales y Sistemas de Información ofertado por la UPV. En él se pretende aprovechar la oportunidad para ampliar los conocimientos adquiridos en el curso adentrándose en la práctica de la programación para dispositivos móviles, la inteligencia ambiental y los servicios web.

También, a lo largo de la exposición del trabajo, se presentarán y discutirán conceptos y tecnologías relativamente nuevos en el campo de la computación, tales como la Arquitectura Orientada a Servicios (SOA), servicios REST, tecnología NFC o los sistemas operativos de los dispositivos móviles.

Para llevar a cabo el desarrollo del proyecto y poder profundizar en los conceptos y tecnologías anteriormente mencionados se propone la implementación de un prototipo adecuado a tales fines.

1.1 Motivación

Hoy en día existen en el mundo millones de teléfonos inteligentes (smartphones) con una capacidad de procesamiento, duración de la batería y conectividad a internet suficientes como para que se hayan convertido en un elemento imprescindible en la vida cotidiana de las personas del primer mundo. El hecho de la ubicuidad y las capacidades computacionales de estos dispositivos junto con la gran cantidad de sensores y tecnologías accesorias que disponen, permiten la creación de nuevas funcionalidades a los usuarios convirtiéndolas en nuevas necesidades. Estas funcionalidades deben de ser aprovechadas tanto por empresas como por administraciones para ofrecer servicios de calidad y valor añadido al usuario final. Todo ello nos lleva a plantear nuevos modelos de negocio, o adecuar a las nuevas tecnologías modelos de negocio ya existentes.

Por otro lado existe una tecnología de comunicación bastante reciente llamada NFC que por sus cualidades muestra un gran potencial junto al uso de teléfonos inteligentes para aportar valor añadido y nuevas funcionalidades. Esta tecnología esta basada en comunicación de corto alcance, apenas 4 o 5 centímetros, lo que invita a pensar que existe una intencionalidad consciente cuando ésta se produce entre dos dispositivos. Además el corto alcance también asegura un nivel de seguridad muy aceptable, ya que es muy difícil que en primer lugar se produzca la comunicación sin intencionalidad y en segundo lugar que existan intrusos que intercepten los datos de la comunicación en sí. Es por ello que se trabaja hoy en día en la implantación de esta tecnología para tareas tan sensibles e importantes como el pago, la identificación personal, el pase a zonas restringidas, canjeo de descuentos, transferencia de archivos, etc. Hablaremos de esta tecnología más adelante, dedicando un capítulo propio para ello.

Por último, existe una reciente arquitectura software basada en servicios, llamada SOA (Service Oriented Architecture). Esta arquitectura define el uso de servicios como soporte a los requisitos del negocio. El principal objetivo es alcanzar el mínimo acoplamiento entre agentes software.

Una arquitectura orientada a servicios es una solución software que pretende permitir a la empresa organizar y hacer uso de múltiples procesos. Con SOA, las aplicaciones software ya no son enormes bloques de funciones y procesos. En cambios, estas aplicaciones se componen de servicios modulares ensamblados. Recordemos que un servicio es una función software simple. Puede ser ejecutada bajo demanda por cualquier sistema, sin tener en cuenta el sistema operativo, plataforma, lenguaje de programación o posición geográfica. Este hecho es clave cuando hablamos de dispositivos móviles, ya que los dispositivos móviles son ubicuos. Además existen gran cantidad de sistemas operativos móviles cuya capacidad de almacenamiento y procesamiento es limitada, y si tratamos de ofrecer ciertos servicios a través de estos dispositivos, SOA se convierte en la opción ideal por su alto desacoplamiento de plataforma y su capacidad de proveer de lógica de negocio y almacenamiento de una forma ubicua a través de la red.

La implementación se SOA se basa en servicios web. Hoy en día existen dos grandes tipos de servicios web: los servicios web al uso (Web services) que luego comentaremos en detalle, y los servicios REST. Estos últimos son los más utilizados en dispositivos móviles por su sencillez, versatilidad y funcionalidad.

En la línea del aprovechamiento de las nuevas tecnologías, recientemente mencionadas, en modelos de negocios existentes, se presenta esta tesis, que se acompaña de un prototipo adecuado a tales circunstancias intentando aportar valor añadido en un supuesto concreto.

1.2 Objetivos

Los objetivos del proyecto son profundizar en tecnologías modernas y así complementar la formación recibida en el máster de un modo mucho más práctico, de cara a un desarrollo profesional de futuro.

Es por ello que se eligen los sistemas operativos móviles, la tecnología NFC y los servicios web como base del proyecto para tratar de unirlos mediante un fin común a través de un prototipo que pueda ser enmarcado en una propuesta de negocio viable, coherente y de futuro, pero también de presente, es decir, que hoy en día pudiera ser implementada y llevada a cabo sin más problemas o hándicaps tecnológicos.

La formación recibida en el máster sobre dispositivos móviles es escasa. En cierto modo es un hecho totalmente entendible debido a la imposibilidad de abarcar todos los aspectos y tecnologías que hoy en día ofrece el mundo de la informática. Para paliar esta manca de formación por mi parte y poder llevar a cabo el proyecto sin problemas, se realizaron varios cursos de formación además de proceder a una documentación exhaustiva sobre el tema. En concreto se realizó un curso de 200 horas sobre programación en Android, y un curso de Especialista Universitario en Computación Móvil y Ubicua de 30 créditos.

Lo mismo sucede en cuanto a la formación sobre la tecnología NFC. El objetivo era dominar esta tecnología hasta el punto de poder desarrollar un prototipo práctico que aglutinará gran parte de sus capacidades. Sin embargo, al contrario que sucede con el desarrollo en sistemas operativos móviles, la capacidad de recibir formación específica en este sentido es muy reducida, por no decir nula. Por tanto se ha tenido que realizar un gran trabajo de documentación y de investigación de los recursos existentes en la red sobre dicho fin para poder conseguir este objetivo.

En cuanto a SOA y REST, la formación en el master ha sido bastante más amplia, dedicando una asignatura específica para ello. Por tanto el conocimiento de base al inicio del proyecto era mucho mayor que en los casos anteriores, aunque existía una manca práctica que fue resuelta del mismo modo que en el apartado de sistemas operativos móviles.

Una vez realizado el prototipo se darán por conseguidos los objetivos de este proyecto, pero antes que nada habrá que definir un prototipo adecuado a tales fines.

En el siguiente apartado se muestra una vista rápida del prototipo pretendido, con la intención de que el lector tenga una idea mucho más clara del mismo antes de adentrarse en profundidad en los fundamentos y desarrollo de esta tesis de máster.

1.3 Introducción al prototipo: BeepVip

Antes de entrar en profundidad en el desarrollo del proyecto, veamos un esquema básico del prototipo a realizar, con la intención de demostrar que la motivación y objetivos del proyecto se alinean perfectamente con el prototipo elegido. Para ello presentemos un boceto de la arquitectura básica del prototipo, que a partir de ahora llamaremos BeepVip.

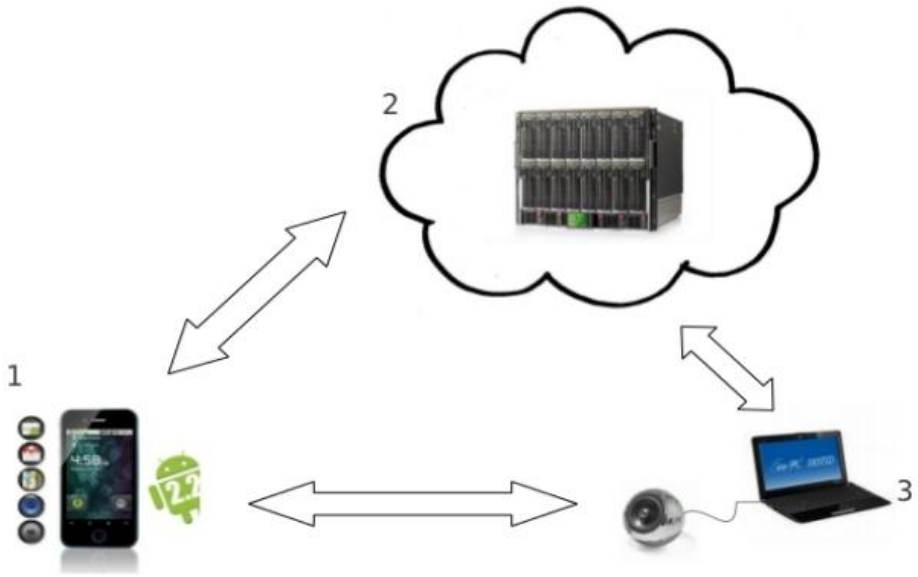


Figura 1. Boceto de arquitectura básica BeepVip

La propuesta de arquitectura básica del proyecto constará de un aplicación móvil (1) en la que usuario podrá consultar las ofertas o tickets disponibles para posteriormente adquirirlas. Una vez adquiridas podrá canjearlas mediante técnología NFC en el local pertinente. El local dispondrá de un software específico para tal fin junto con los dispositivos necesarios para el reconocimiento de la información ofrecida por el dispositivo del usuario, como es un reader NFC (3). La arquitectura se completa con un conjunto de servicios web (2) que sirven y almacenan la información adecuada tanto al software del usuario como al del local.

Con esto ya tenemos una visión general del proyecto, y así poder entrar a tratar cada uno de los temas relevantes relacionados con mucho más detalle.

1.4 Estructura de la tesis

Con este último apartado se de por concluido el primer capítulo de este documento. En esta sección se expondrá la estructura restante del documento para dar una idea mas clara al lector.

En el segundo capítulo (Planteamiento del problema) se trata de dar una visión de conjunto de la situación actual del mercado en el cual se pretende instaurar el prototipo, con la intención de definirlo de tal modo que aporte valor añadido suficiente como para convertirse en un prototipo de recorrido empresarial.

En los capítulos 3, 4 y 5 se aborda toda la parte teórica respecto de los conceptos y tecnologías a usar en el proyecto. Por tanto se discutirán los fundamentos de NFC, los sistemas operativos móviles (en especial Android), SOA y servicios REST.

.

El capítulo sexto recoge la parte más práctica del documento, ya que es en el que se expone el desarrollo del prototipo asociado a la tesis. Se entrará en el análisis previo, la elección de tecnologías específicas, APIs, desarrollo de la solución, etc. Este capitulo se completa con un apartado donde se proponen las mejoras a introducir en el prototipo.

Por último, en el capítulo 7, se llevarán a cabo las conclusiones que se estimen oportunas sobre el proyecto, abarcando múltiples aspectos.

2 Planteamiento del problema

2.1 Introducción

Este proyecto se engloba dentro de la especialidad de sistemas de información del máster al que pertenece, y a su vez se enmarca dentro del enfoque denominado de orientación profesional. Esto hecho lleva consigo la condición de que el proyecto trate sobre trabajos derivados de la actividad profesional, o bien sobre supuestos que puedan ser susceptibles de formar parte de un proyecto empresarial de presente.

En este caso se trata de formular un supuesto empresarial a modo de *startup* que pudiera tener cabida en el mundo empresarial actual. Para ello se intenta desarrollar una idea de negocio suficientemente atractiva y novedosa, mediante el uso de nuevas tecnologías de información y comunicación, que aporte valor añadido en los modelos de negocio concernientes al ámbito de las ofertas de locales, tickets, flyers, etc.

Antes de aportar las líneas generales de esta solución, se deben identificar los principales problemas o limitaciones que cada modelo de negocio predominante pudiera tener, desde un punto de vista tanto de usuario final como de empresa, o incluso social.

Para ello, inicialmente, definiremos cada modelo de negocio a través de sus características principales e indagaremos en sus posibles defectos o limitaciones.

2.2 Modelos de negocio existentes

Existen diversos modelos de negocio en el ámbito sobre el cual quiere definirse una solución de valor añadido en el sector. De todos los existentes se seleccionan los que se consideran más importantes en cuanto a presencia: Modelo de venta de tickets/entradas, modelo de ofertas exclusivas y modelo de flyers. Hay que notar que existen modelos de negocio mixtos, en los que se combinan los fundamentos de uno o varios modelos tipo para llevar a cabo el servicio.

2.2.1 Modelo de venta directa de tickets/entradas

Este modelo es quizá el menos novedoso y más arraigado de todos. Es un modelo mediante el cual es usuario o cliente accede de forma activa a la compra de un determinado ticket o entrada. Generalmente el usuario debe de conocer de antemano la existencia del evento en cuestión mediante los diversos medios de comunicación existentes.

Un ejemplo de este tipo de modelo sería la compra en taquilla de una entrada de cine o de un concierto.

En este tipo de ejemplo se pueden observar, algunos inconvenientes, tales como:

- *Proactividad del usuario:* El usuario debe de mostrar un interés previo, y mediante los recursos de información disponibles a su alcance enterarse de la fecha, hora, lugar, etc. de la venta. Por otro lado, este hecho hace que la empresa tenga que gastar recursos en poner al alcance del cliente potencial toda la información necesaria, ya sea mediante publicidad o de cualquier otro modo.

- Limitación espacio-temporal: Al tratarse de una venta "física" existe una limitación del horario de venta al público, así como de la ubicación de la venta, forzando al usuario a desplazarse para poder adquirir el ticket o entrada. Este hecho también limita a la empresa que puede perder clientes potenciales por no ajustarse el horario o el lugar a sus posibilidades.

- Empleabilidad: Este tipo de modelo exige a la empresa tener que contratar a empleados dedicados exclusivamente a tal fin, aumentando los costes de prestación del servicio, que finalmente repercuten en el cliente. Por otro lado hay que decir que socialmente considerar este punto como un inconveniente puede generar controversia.

- Ecología: Los tickets o entradas suelen ser físicas, lo que implica un gasto innecesario de papel y tinta. Ello genera un aumento de la deforestación y contaminación del planeta, lo que es un problema social muy a tener en cuenta. También genera un aumento de costes a la empresa.

Actualmente algunos de estos inconvenientes han sido mejorados o incluso anulados con la incursión de nuevas tecnologías, sobretodo en cuanto a la limitación espacio-temporal y la empleabilidad.

Un ejemplo claro puede ser la compra de entradas mediante el servicio ServiCaixa de La Caixa que permite sacar las entradas desde un cajero de banca disponible todo el día. Otro ejemplo puede ser www.ticketmaster.es en el que la compra de entradas se realiza mediante internet.

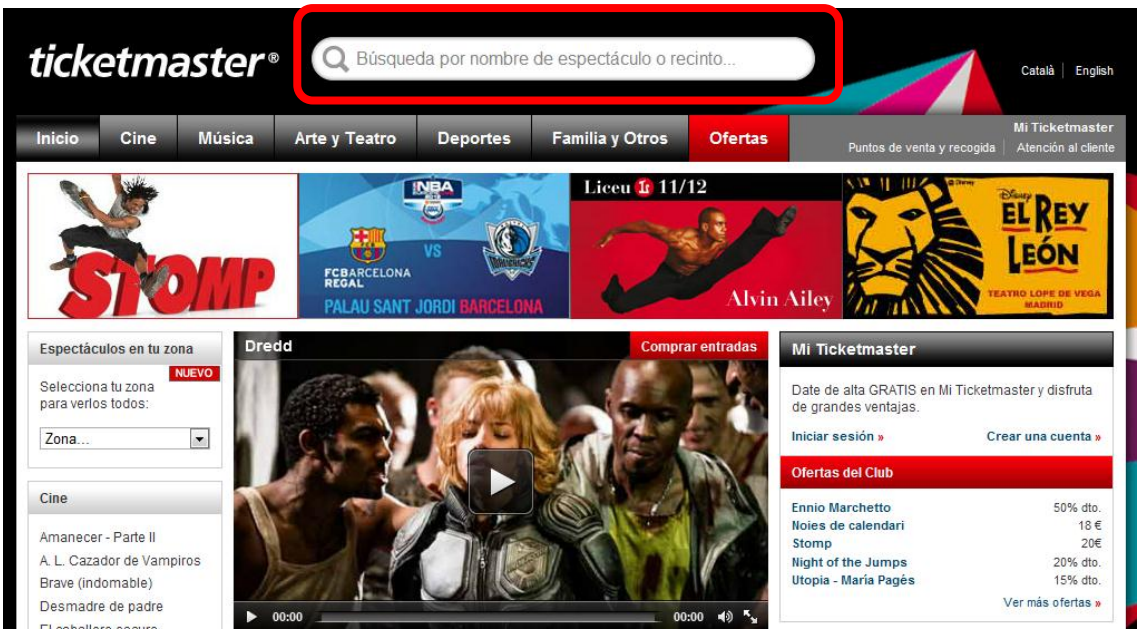


Figura 2. Ejemplo de proactividad del usuario del modelo venta tickets/entradas

Aun así, estos ejemplos no solucionan los inconvenientes de la proactividad y la ecología.

2.2.2 Modelo de ofertas exclusivas

Este modelo esta teniendo un gran éxito últimamente. Se trata de plataformas online que ofrecen grandes ofertas a sus clientes registrados. Relacionándolo con el modelo anteriormente descrito, se puede observar que este modelo mejora sustancialmente el inconveniente de la proactividad. Es decir, el cliente recibe buenas ofertas fácilmente adquiribles de una forma pasiva, ya que el evento o actividad es propuesto por la empresa.

Entre este tipo de empresas podemos destacar Groupon, Groupalia, Planeo, etc. Sin embargo mantiene el problema de la ecología ya que generalmente el cliente debe imprimir el ticket o cupón, aunque libera del gasto de impresión a la empresa que ofrece el servicio ofertado en este tipo de plataformas. Los inconvenientes más importantes de este tipo de modelo, son:

- Cantidad de ofertas: Normalmente se ofrecen poca cantidad de ofertas al usuario, aunque las ofertas suelen ser muy buenas en relación calidad-precio.

- Escasa limitación de ofrecimiento: Aunque no lo parezca, este es un problema capital. El problema es que esas grandes ofertas se ofrecen a una gran cantidad de usuarios sin apenas criterio, lo que implica que sea ofertada a una gran cantidad de clientes no potenciales que aprovechan la oportunidad.

  Por ejemplo, si una marisquería oferta una mariscada a 15€ por persona y un grupo de adolescentes adquieren los cupones para celebrar su cumpleaños, la marisquería habrá invertido dinero y tiempo en hacerse conocer entre clientes potenciales mediante estas plataformas, mientras que lo que ha recibido no son solo clientes no potenciales sino que además pueden dañar la imagen del negocio.

- Escasa filtración de ofertas: Esta es una de las consecuencias del punto anterior, pero esta vez visto desde el punto de vista del usuario. Al existir poca limitación y filtración de ofrecimiento, el usuario se encuentra que la gran mayoría de las ofertas no le interesa en absoluto, haciéndole perder su valioso tiempo.

Por tanto, se deduce de todo lo anterior que un modelo que evite el inconveniente del usuario proactivo debe de procurar que las comunicaciones entre empresa ofertante y cliente, se realicen exclusivamente entre clientes potenciales, por el bien de ambas partes. Siendo así, se garantiza la sinergia necesaria para hacer que un negocio sea sostenible en el tiempo.

2.2.3 Modelo de flyers

Este modelo es un modelo de ofertas es un modelo típico en locales de ocio nocturno. Ofrece al cliente potencial lo que está buscando de forma pasiva para éste, ya que la figura del relaciones públicas busca clientes potenciales por las calles y bares cercanos al local para ofrecer los descuentos. Sin embargo, aunque en cuanto al inconveniente de proactividad es quizá el mejor de todos, no soluciona el problema espacio-temporal, ni el de empleabilidad, ni el ecológico.

2.2.4 Conclusión

Se han expuesto diversos modelos existentes en el ámbito de la venta de tickets, entradas, etc. sin encontrar el modelo perfecto según criterios de proactividad, empleabilidad, disponibilidad y ecológicos. La pregunta es sencilla, ¿Es posible crear un nuevo modelo capaz de solucionar todos estos inconvenientes teniendo en cuenta el beneficio del usuario final y de las empresas ofertantes de servicios?

El inconveniente de la proactividad se soluciona a través de algún tipo de plataforma que ofrezca descuentos a clientes potenciales. Para ello es conveniente filtrar al máximo los descuentos/tickets/entradas según criterios de sexo, edad, localización, hábitos de consumo, etc.

La empleabilidad tiene la solución a través de la externalización y automatización de servicios. Si se externaliza determinado servicio a una empresa especialidad en ofrecer dicho servicio, por norma general, se reducen los costes, aunque se pierde cierto control en dicho servicio o funcionalidad. La automatización versa en la misma dirección de reducción de costes, pero por contra ofrece mayor control al poder monitorizar el proceso.

En cuanto a la disponibilidad, es evidente que a través de servicios automatizados de índole tecnológico (webs, aplicaciones, etc.) se consigue una mayor disponibilidad del servicio, ya que el servicio es ofrecido por una o varias máquinas o elementos software capaces de estar disponibles prácticamente todo el tiempo. La otra dimensión de la disponibilidad es la espacial, es decir, que no importe el lugar en el que se encuentre la persona interesada en

cierto ticket para que ésta pueda adquirirlo en el momento. Este problema solo puede ser solucionado a través de dispositivos ubicuos con capacidad de realizar el servicio.

Por último, la cuestión ecológica puede ser subsanada a través de la virtualización de los elementos físicos como entradas, descuentos, flyers, etc.

Por tanto, se concluye que las nuevas tecnologías de la información y comunicación son capaces de evitar casi todos los inconvenientes que pueden surgir en este tipo de modelos de negocio, así como en muchos otros. Una solución de vanguardia tecnológica parece el camino adecuado hacia el éxito. Definamos pues dicha solución para el caso que nos ocupa.

2.3 Definición de la idea: BeepVip

La solución más adecuada pasa por intentar evitar los cuatro inconvenientes anteriormente comentados: proactividad del usuario, disponibilidad, empleabilidad y ecología. Además se pretende dar ciertos servicios de valor añadido.

La solución propuesta consta de una aplicación para smartphones, que llevará por nombre BeepVip. Para explicar el funcionamiento y los servicios que se pretenden llevar a cabo, se cree conveniente aportar los casos de uso desde dos perspectivas distintas: los usuarios y los locales o empresas que ofrecen los tickets a sus clientes potenciales (los usuarios).

Figura 3. Logo de BeepVip

2.3.1 La visión del usuario

El usuario se descarga la aplicación del market de su SO móvil (disponible sólo en smartphones) y se dá de alta aportando un nick, contraseña, edad, sexo y código postal. Se

podría utilizar el servicio de Facebook para usar las mismas credenciales reduciendo los datos a aportar.

Una vez está dado de alta, en la aplicación puede consultar las ofertas disponibles haciendo uso de varios filtros. El primer gran filtro es la división por módulos. La aplicación tiene una interfaz principal en la que se encuentran los elementos comunes a todos los módulos y los módulos en sí. Un módulo corresponde a un tipo de ticket, por ejemplo el módulo "Fevernight" corresponde a descuentos de locales de ocio nocturno, y el módulo "Cinema" corresponde a entradas a salas de cine. Los módulos son configurables por el usuario, es decir, el usuario decide los módulos de los que quiere disponer a través de opciones de configuración. Cuando entra en uno de los módulos puede consultar los descuentos/entradas disponibles de forma filtrada. Por ejemplo podría ver las ofertas de ocio nocturno que se le han hecho para un día en concreto, para un tipo de local concreto, en una zona determinada, las destacadas, etc.

Figura 4. Boceto interfaz principal BeepVip

Al consultar una oferta puede ver lo que se ofrece, el precio, donde se ubica el local, opiniones del local, la película, etc., cuantos hombres y cuantas mujeres han adquirido la oferta, etc. y la puede descargar o incluso compartirla con determinados amigos que pueden estar interesados.

Una vez el usuario ha adquirido la oferta, puede gestionarla (borrar, compartir, canjear) desde una interfaz accesible desde la interfaz principal en la que se encuentran todos los tickets disponibles descargados por el usuario se dispone a canjearla. Cuando se dispone a canjearla, al entrar al local hace check-in mediante tecnología NFC con su smartphone en el dispositivo de check-in del local. Así de una manera automática la persona encargada d

el cobro en el local sabe que oferta dispone el usuario y puede cobrarle (incluso se podría cobrar automáticamente y de forma transparente al usuario, asociando BeepVip con una cuenta de banco o PayPal por ejemplo). En eso momento el check producido pasa a la base de datos de BeepVip, y desaparece de los tickets disponibles en la aplicación del usuario.

2.3.2 La visión del local

Cuando el local contrata los servicios de BeepVip dispone de varios servicios para que la tarea de hacer llegar sus productos o servicios al cliente potencial sea más eficiente, y por tanto poder reducir costes.

Un local que quiera hacer un descuento o vender una entrada o determinado servicio, en definitiva, un local que quiera ofrecer un ticket en la plataforma deberá entrar a la web de BeepVip, loggearse como cliente y rellenar el formulario para crear el ticket. Los tickets son personalizados, es decir, el local que ofrece el ticket puede hacer ofertas selectivas a hombres o mujeres, ofertas dependiendo de la edad, dependientes de la zona geográfica de residencia e incluso según los hábitos de consumo mostrados a lo largo del uso de la app (así por ejemplo una discoteca puede premiar la fidelidad de un usuario, o hacer ofertas suculentas a clientes potenciales), y todo a través de una correcta cumplimentación del formulario. De este hecho se desprende que BeepVip dispone de ofertas distintas para cada usuario.

Además existen lo que se llaman ofertas VIP. Este tipo de ofertas llegar a través de una notificación a la aplicación del usuario, reclamando su atención y haciendo valer su exclusividad.

En todo momento puede consultar a través de la web el estado de las ofertas vigentes o del histórico de ofertas, por ejemplo el alcance de la oferta (numero de usuarios BeepVip a los que llega la oferta) o el número de descargas que se han producido hasta el momento.

BeepVip provee a sus clientes del dispositivo de check-in necesario para poder canjear los tickets en el local. Cuando un usuario se dispone a canjear su ticket debe de hacer uso de la tecnología NFC en el dispositivo de check-in del local. De este modo el local sabe el tipo de descuento que dispone el usuario y procede al cobro y la dotación del servicio.

El local paga por la prestación de los servicio de BeepVip de la siguiente manera:

- Una cuota mensual por la provisión del servicio, el alquiler del dispositivo de check-in, soporte, la posibilidad de hacer un número limitado de ofertas, etc.

- Si se quieren hacer más ofertas de las estipuladas por mes, se debe de pagar por ello.

- BeepVip cobra a los locales por casos de éxito, es decir, se cobra una determinada cantidad por cada ticket canjeado. De esta forma hay relación muy directa entre el beneficio del local y el pago de los servicio a BeepVip, cuanto más suba la factura a emitir, más provecho ha sacado el local de la plataforma.

- Por último, existe la posibilidad de recibir informes personalizados de la información que BeepVip dispone sobre los hábitos de consumo de sus usuarios. Este tipo de informes también se pagan.

2.4 Modelo de negocio

En el apartado anterior ya se ha hablado bastante sobre el modelo de negocio de BeepVip al intentar describir los servicios y funcionalidades de la plataforma. Sin embargo, para tener una idea mucho más precisa de su modelo de negocio y entender la propuesta planteada es necesario abarcar todas las dimensiones del negocio.

Entrar de lleno en el modelo de negocio es una tarea que se puede abordar de varias maneras. En este caso se ha optado por basarse en el denominado “lienzo del negocio” analizando cada una de las partes que lo componen para el caso en concreto.

A la hora de abordar el lienzo del negocio de BeepVip se cree necesario simplificar el modelo con la intención de hacerlo mucho más entendible para el lector. Por tanto en lugar de presentar un lienzo de todo el negocio, se cree oportuno presentar el lienzo de un solo módulo de entre los diversos módulos existentes en BeepVip, ya que los demás módulos que componen la aplicación son en esencia muy parecidos. El módulo elegido para la ocasión es FeverNight que abarca el supuesto de descuentos y entradas en locales de ocio nocturno.

2.5 Lienzo del negocio

Pasemos ahora a analizar cada elemento del lienzo de negocio que se propone para este caso, y que se anexa al documento. Se puede considerar cada módulo de BeepVip como una app independiente a la hora de analizarla, ya que el sistema modular propuesto para la plataforma une los elementos comunes entre módulos, pero abarca las particularidades de cada módulo de una forma independiente del resto, con el objetivo de dotar de mayor usabilidad a la app.

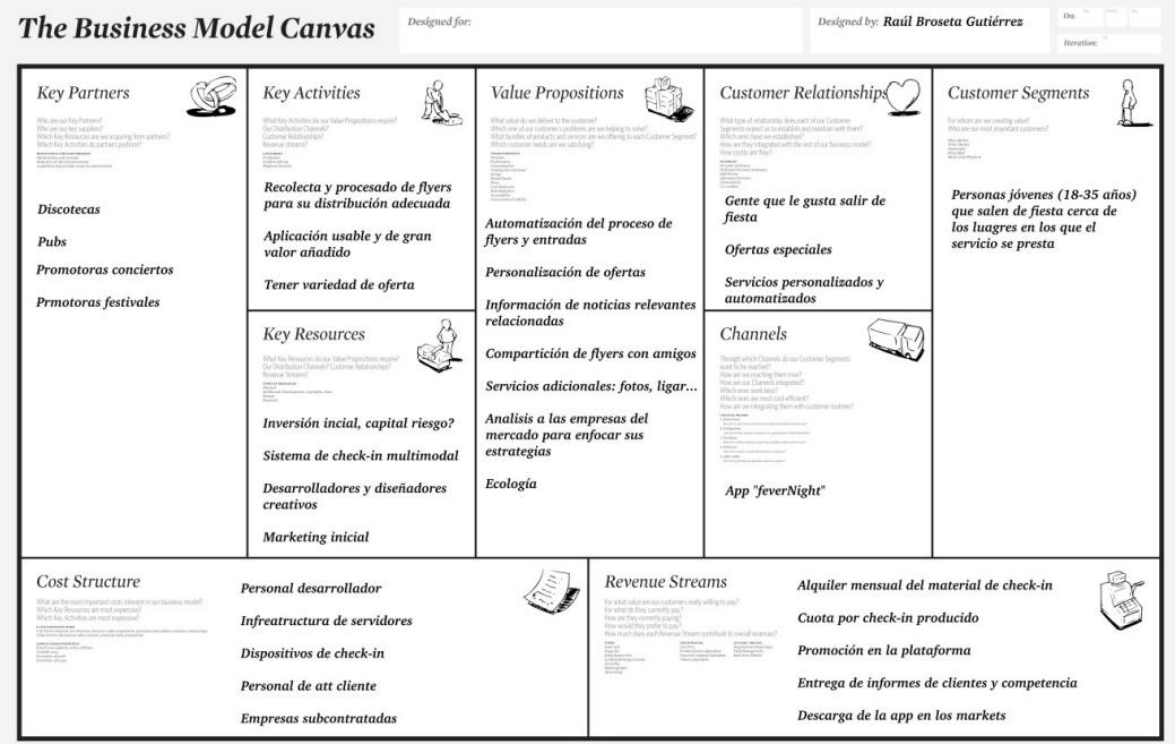


Figura 5. Lienzo negocio módulo Fevernight

Key partners: Evidentemente los socios clave para poder desarrollar la idea de negocio son básicamente las empresas que disponen de locales de ocio nocturno (discotecas, pubs, etc..). Sin ellas el modelo sería imposible de implantar, por lo que es necesario poder aportar mucho valor añadido a las empresas para que sea factible la colaboración y se produzca la sinergia adecuada. No hay que olvidar que el grueso de los ingresos de esta idea de negocio pasa por el cobro de los servicios prestados a los locales, mediante una pequeña cuota de alquiler de material y oferimiento del servicio (soporte, presencia en la plataforma...) y sobretodo mediante el cobro por cada caso de éxito producido (es decir, por cada usuario que ha utilizado feverNight para acceder a su local).

Key Activities: Las actividades clave de la empresa deben ser la consecución de un producto (app) que tenga un diseño atractivo a los clientes y gran usabilidad. Por otro lado se ha de garantizar la correcta distribución de las ofertas, adecuadas a los clientes potenciales, a través de gestión de servicios y servidores web así como técnicas de minería de datos. Finalmente es una actividad principal de la empresa conseguir el mayor número de empresas de locales de ocio nocturno asociadas con feverNight.

Key Resources: Los recursos claves propuestos pasan por obtener en primer lugar la financiación inicial necesaria para poner el negocio en marcha, al poder ser a través de fondos de capital riesgo. Otros recursos muy importantes son los recursos humanos que formarían el equipo de BeepVip, es decir, desarrolladores, diseñadores y comerciales. También hay que tener en cuenta como recurso clave la disposición de un dispositivo de check-in multimodal que permita hacer los check-ins a los usuarios según las características de su smartphone, teniendo como visión de futuro una convergencia a la tecnología NFC. Este dispositivo es vital. Por último se considera como recurso clave el marketing inicial para hacer que los usuarios potenciales hagan uso de la app.

Value propositions: Como propuestas de valor hacia los usuarios potenciales destacaremos la automatización y personalización de la adquisición de ofertas, así como otras funcionalidades de la plataforma que complementan la función principal, como por ejemplo compartir ofertas e invitaciones. Como propuestas de valor a las empresas asociadas se puede destacar que la plataforma hace que la comunicación con los usuarios potenciales sea más focalizada y mejor, además de reducir costes en cuanto a relaciones públicas, impresión de ofertas (también es más ecologico), gastos de publicidad, etc. Por último también se pueden ofrecer servicios de pago de análisis sobre los datos obtenidos por la plataforma para que la empresa pueda hacer un análisis del mercado y preparar su estrategia de negocio con más acierto.

Customer Relationships: La relación con los clientes es una relación basada en la prestación de un servicio gratuito que les permite obtener ofertas personalizadas que van más allá de las posibles plataformas de compras por impulso a través de ofertas (ej. Groupon) que pueden utilizar en sus planes de ocio nocturno.

Channels: El canal obviamente es una aplicación de Smartphone.

Customer Segments: El segmento de mercado está mu bien definido, gente joven que le gusta salir de fiesta en las ciudades en las que el servicio está disponible.

Cost Structure: Los costes de la BeepVip vienen determinados en primer lugar por sus recursos humanos (empleados). Además hay que añadir los costes de mantenimiento del servicio como servidores y dispositivos de check-in. Y por último los servicios prestados por otras empresas, por ejemplo los comerciales subcontratados para la campaña de marketing inicial.

Revenue Streams: Este apartado es el determinante en cuanto a la sostenibilidad del negocio en sí. Los ingresos de feverNight vendrán por el cobro de una cuota mensual a las empresas de locales de ocio nocturno por el alquiler del dispositivo de check-in, el soporte técnico y la posibilidad de ofrecer en la plataforma un determinado número de ofertas. Si se quieren hacer más ofertas o que éstas sean destacadas se tendrá que hacer efectivo otro pago al respecto. Pero principalmente, los ingresos van estrechamente relacionados con los casos de éxito, es decir, feverNight cobraría a sus socios una cuota de por ejemplo 1€ por cada persona que ha realizado un check-in. También existe la posibilidad de obtener ingresos extra a través de la venta de informes a los locales, o una vez implantada la plataforma socialmente cobrar por descarga en el market al usuario final.

3 NFC

3.1 Introducción

NFC (Near Field Communication) es un conjunto de estándares principalmente utilizados por smartphones y dispositivos similares para establecer comunicación entre ellos e intercambiar datos, ya bien sea de forma unidireccional o bidireccional. La principal característica de esta tecnología se centra en una radio-comunicación de alta frecuencia y muy corto alcance, no más de unos 5 centímetros. Dichos estándares cubren todos los protocolos de comunicación y formatos de intercambio necesarios para una correcta comunicación entre dispositivos. Éstos están basados en la tecnología RFID (identificación por radiofrecuencia), incluyendo el estándar ISO 14443 y FeliCa, por ello se puede considerar a NFC un caso particular o extensión de RFID. Los estándares incluyen el ISO 18092 (aprobado en 2003) y los propios definidos por el NFC Forum, organismo formado en 2004 por Nokia, Philips y Sony para la correcta estandarización e implantación de la tecnología NFC. Hoy en día el NFC Forum cuenta con más de 160 miembros.

Aunque en un principio el corto alcance de la tecnología NFC pueda parecer un inconveniente frente a tecnologías similares como RFID o BlueTooth, hay que señalar que es justamente esta característica la que le dota de un gran valor añadido frente a las tecnologías ya citadas. El hecho de que la comunicación no pueda producirse a más de 5 centímetros "garantiza" la seguridad de la comunicación al ser prácticamente imposible interceptar la señal sin que una persona se percate de ello. Además si dos dispositivos establecen una comunicación NFC podemos asegurar que se encuentran sumamente cercanos. Estas consecuencias lógicas de la propia definición de NFC la hacen especialmente adecuada para usos humanos en los que se puede ver afectada la privacidad de las personas, como por ejemplo el intercambio seguro de archivos, pagos por móvil, identificación, etc.

Como ya se ha dicho, la tecnología NFC tiene un gran potencial en el mundo actual y por sus propias características puede ser usada en una gran variedad de escenarios. Suele asociarse NFC a los pagos por móvil debido al gran empeño de grandes compañías por hacerse con ese mercado y la gran revolución que ello supondría, pero éste no es más que un ejemplo de las muchas posibilidades que alberga. Así pues, la tecnología NFC puede estar y estará

presente en tareas tan variadas como identificación de personas, etiquetado de objetos, pagos por móvil, canjeo de tickets, transporte, loggeo seguro, accesos, intercambio de archivos, intercambio de tarjetas de visita, agregar personas a redes sociales, etc. En la siguiente figura podemos ver un breve esquema de ello.

Figura 6. Esquema de posibilidades NFC

Sin embargo, hoy en día, es una tecnología poco implantada en la sociedad, en cierto modo debido a que es relativamente nueva. Ya existen sistemas operativos móviles como Android que ofrecen un gran soporte, y los demás están en ello, por lo que es de esperar que no tarde en eclosionar socialmente.

3.2 Historia

La tecnología NFC es una de las tecnologías más modernas de nuestra época, por tanto su historia es bastante escueta, pero es necesario conocer cuáles han sido sus primeros pasos. La historia cronológica de NFC a grandes rasgos es:

- 1983: Charles Walton registra la primera patente asociada con RFID
- 2003: Se lleva a cabo el primer estándar 18092 entorno a NFC, como caso particular de la tecnología RFID
- 2004: Nokia, Philips y Sony crean el NFC Forum, organismo encargado de su estandarización, implantación y certificación
- 2006: Primeras especificaciones para tags NFC

- 2006: Primer móvil con tecnología NFC (Nokia 6131)
- 2009: El NFC Forum publica el primer estándar peer-to-peer para poder transferir contactos, URLs, etc.
- 2010: Sale el primer Android con NFC (Samsung Nexus S)
- 2010: Google demuestra cómo usar NFC en Android en el Google IO
- 2010: Symbian da soporte a NFC (Anna)
- 2010: Android da soporte a NFC (API 9). Sólo tags.
- 2011: Android soporta NFC entre dispositivos (Beamming)
- 2011: RIM es la primera compañía en ser certificada por MaterCard para el pago por móvil

3.3 Fundamentos

3.3.1 RFID

Como antes se ha comentado, NFC se pude considerar un caso particular de RFID, y por tanto basa su fundamento en esta tecnología. RFID son las siglas de Radio Frecuency IDentification (Identificación por Radiofrecuencia). Los sistemas de identificación por radiofrecuencia están destinados a la identificación de objetos a distancia sin necesidad de contacto, ni siquiera visual. Para ello, se requiere de ciertos elementos como etiquetas o tags RFID, consistentes en un microchip y una antena de radio de reducidas dimensiones y que sirve para identificar unívocamente al elemento portador de la etiqueta.

La información almacenada en las etiquetas RFID puede ir desde 1 bit hasta varios kilobytes, siendo la capacidad máxima existente de 4KB y dependiendo principalmente del sistema de almacenamiento que posea el tag RFID.

Por otra parte es necesario un lector capaz de obtener los datos almacenados en las etiquetas RFID. Lo normal es tener un dispositivo que tenga una o varias antenas que emitan continuamente ondas de radio y que reciban las señales devueltas por las etiquetas RFID.

El propósito inicial y en el que se fundamenta la tecnología RFID es el de transmitir la identidad de un objeto, similar a un número de serie único, mediante ondas de radio.

El origen de la tecnología RFID está relacionado con la Segunda Guerra Mundial, en la que el uso del radar permitía la detección de aviones a varios kilómetros de distancia, aunque no su identificación. Los sistemas de radar y de comunicaciones por radiofrecuencia sufrieron avances relevantes en las décadas de los años 50 y los 60, años en que los científicos de los países más avanzados trabajaban para explicar cómo identificar objetos remotamente. Fruto de numerosas investigaciones en este campo son la creación de patentes para dispositivos RFID.

Las primeras patentes para dispositivos RFID fueron solicitadas en EEUU, en el año 1973, cuando Mario W. Cardullo se presentó con una etiqueta RFID activa que portaba una memoria que podía ser rescrita. Ese mismo año, Charles Walton recibió la patente para un sistema RFID pasivo que abría puertas sin necesidad de llaves. Una tarjeta con un tag comunicaba una señal al lector de la puerta, que desbloqueaba la cerradura tras validar la tarjeta.

Ya entre los años 70 y 80 tienen lugar las primeras implementaciones comerciales de RFID. Sin embargo, todos los proyectos en torno a esta tecnología eran propietarios y era difícil que se interrelacionaran. Además, los costes eran muy altos y por lo tanto, el uso generalizado de la tecnología RFID estaba aún lejano.

Es en la década de los 90 cuando se producen importantes avances en la ciencia de materiales, avances que permitirán bajar de manera drástica el precio de las etiquetas RFID. Los organismos internacionales inician esfuerzos para desarrollar estándares de etiquetas.

3.3.1.1 Etiquetas RFID. Funcionamiento

Con la comercialización de los primeros sistemas RFID, se hace necesario el desarrollo de nuevos tipos de etiquetas RFID, etiquetas que poco a poco están sustituyendo a las etiquetas de códigos de barras y a las tarjetas magnéticas en todas sus aplicaciones.

Mejoras en su capacidad de emisión y recepción han llevado a extender su uso en ámbitos tanto domésticos como de seguridad nacional.

Las etiquetas RFID son el componente principal de cualquier sistema de radiofrecuencia y son las encargadas de emitir una señal de respuesta a otra señal enviada por un lector. Estas etiquetas RFID también reciben el nombre de transponder (Transmitter + Responder).

Figura 7. Tag RFID

Los elementos de una etiqueta o tag RFID son los siguientes:

- *Microantena:* Encargada de recibir las señales enviadas por el lector, responder a dichas señales y suministrar la energía necesaria a la etiqueta (en algunos tipos de etiquetas).

- *Microchip:* Almacena un número de identificación (UID) y contiene la lógica de operación de la etiqueta para una correcta comunicación con el lector. El chip se compone de: radio receptor, radio modulador, control lógico y sistema de energía.

- *Memoria:* Se compone de una parte no volátil denominada ROM, que contiene instrucciones básicas para su funcionamiento, y una memoria RAM para almacenar datos durante la comunicación con el lector.

- *Otros componentes electrónicos:* Procesan la señal de la antena y se encargan de procesar los datos.

Las etiquetas RFID pueden ser de tres tipos dependiendo del lugar del que provenga la energía que utilizan en la comunicación.

- *Etiquetas Pasivas:* Se caracterizan por no incorporar batería, lo que les obliga a extraer la potencia de una fuente externa, en este caso el lector. Cuando el receptor se dispone a realizar la lectura del tag envía energía que activa a la etiqueta para que esta pueda transmitir la información.

  Este tipo de etiquetas poseen unos recursos muy limitados, por lo que pueden almacenar solamente unos pocos KBytes y normalmente son de solo lectura. Su campo de cobertura es también muy reducido (con transmisiones desde pocos centímetros hasta 3 metros). No obstante, su bajo coste hace que sean los tags más utilizados.

- *Etiquetas Activas:* Estas etiquetas RFID llevan incorporada su propia fuente de alimentación y puede enviar señales a los lectores desde más de 100 metros. Además de tener mayor capacidad de almacenamiento en la memoria del tag.

  Las etiquetas activas pueden ser de lectura/escritura o solo lectura, y tienen un mayor coste por chip y son de mayor tamaño que las etiquetas pasivas. Por este motivo, solamente son usados en aplicaciones que realmente los necesiten y no requieran un consumo muy elevado de transmisiones.

- Etiquetas semi-activas o semi-pasivas: Utilizan una batería para activar la circuitería del chip. Sin embargo, la energía para generar la comunicación es la que recoge de las ondas de radio del lector, como ocurre con las etiquetas pasivas. Aunque su respuesta es más rápida en comparación con estas etiquetas pasivas.

Así, en toda etiqueta RFID puede ser almacenada una pequeña cantidad de información. A priori, estos datos estaban destinados a la identificación del objeto portador de la etiqueta. Hoy en día, se puede almacenar cualquier información, provocando que los datos contenidos en dichas etiquetas puedan desencadenar procesos de mayor complejidad.

3.3.1.2 Lectores RFID. Funcionamiento

El lector RFID es el dispositivo encargado de la comunicación con el tag RFID. Puede realizar la lectura del tag o efectuar una lectura/escritura, siendo esta última por proximidad y sin ningún tipo de contacto.

Estos lectores pueden escribir uno o varios tags y su funcionamiento es sencillo. La antena del lector crea un campo magnético y cuando el tag entra en contacto con el campo magnético creado por el lector, reacciona de forma inmediata y envía al lector la información almacenada. El lector decodifica esos datos que por medio de una infraestructura de red son procesados y tratados para efectuar las acciones necesarias.

Figura 8. Reader RFID

Existen dos tipos de lectores o interrogadores diferentes dependiendo de si disponen de bobina simple o doble. En los sistemas con bobina simple, la misma bobina sirve para transmitir la energía y los datos. Son más simples y de menor coste, pero poseen un menor alcance. Por su parte, los sistemas interrogadores de dos bobinas se caracterizan por emplear una de ellas para transmitir energía y la otra para trasmitir los datos. Son más caros, pero tienen mayores prestaciones.

El precio de los lectores es variable y puede oscilar entre unos cien euros hasta unos miles. La velocidad de lectura/escritura también depende mucho del tipo de lector, ofreciendo mayores velocidades los lectores fijos.

Según otro criterio, podemos clasificar a los lectores en función de su portabilidad, diferenciando a los lectores móviles de los lectores fijos.

- *Lectores móviles:* Son dispositivos que pueden transportarse de un lugar a otro gracias a su pequeño tamaño y a la posibilidad de operar con otros dispositivos por medio de una conexión inalámbrica.

- *Lectores fijos:* También denominados de Infraestructura, se instalan de manera permanente dentro del área de acción de los distintos tags a leer/escribir. Estos permiten detectar las señales de los transmisores RFID dentro de un radio de alcance mayor del que permiten los lectores móviles. Estos lectores se utilizan en sistemas de detección y seguimiento de personas y animales en tiempo real.

3.3.1.3 La tecnología MIFARE

MIFARE constituye un estándar tecnológico para comunicaciones de "no contacto" a 13,56MHz cuyo propietario es Philips Electronics. Esta compañía no fabrica etiquetas ni lectores RFID, si no que fabrica y vende al mercado los chips que luego se incluyen en etiquetas y lectores. Esta tecnología permite leer y escribir en etiquetas RFID. A su vez, MIFARE está descrita en el estándar ISO 14443 Tipo A. Las etiquetas MIFARE permiten incrustar un módulo opcional de chip inteligente de contacto. En caso de hacerlo, la etiqueta también cumpliría con el estándar ISO 7816. Pero también está diseñada para tener una banda magnética. En la configuración con banda magnética, la etiqueta también cumple con el estándar ISO 7811.

Las etiquetas sin contacto MIFARE y los lectores de etiquetas MIFARE fueron desarrollados en un principio para transacciones de pago en sistemas de transporte público. Gracias a su corto alcance de lectura, la tecnología MIFARE es especialmente apropiada para realizar funciones de adicción/sustracción. Y aunque las etiquetas inteligentes de contacto también pueden realizar estas funciones, los lectores sin contacto son más rápidos y sencillos de usar y prácticamente no necesitan mantenimiento. Por su parte, las etiquetas sin contacto apenas sufren desgaste.

El alcance típico de lectura/escritura de un lector de etiquetas sin contacto MIFARE es de 2 a 10cm. Y la capacidad normal de memoria de las etiquetas MIFARE es de 1KB de memoria EEPROM, siendo la máxima de 4KB. Una etiqueta MIFARE tiene 16 sectores independientes que pueden configurarse como "monederos" o para el almacenamiento de información general. El primer sector se usa como directorio de la etiqueta, con lo que restan otros 15 segmentos para datos.

Por otra parte, es importante mencionar que la transmisión de datos por radiofrecuencia entre la etiqueta y el lector MIFARE viaja encriptada. Sin embargo, los datos contenidos en la etiqueta no están encriptados, aunque el acceso a los mismos está protegido por una clave de 48 bits. Se realiza una autenticación mutua entre la etiqueta y el lector, esto es, se genera un número aleatorio y de acuerdo con las claves, se envía un mensaje desde la etiqueta al lector. Acto seguido, el lector envía un mensaje a la etiqueta. Esta operación se realiza tres veces para verificar que la etiqueta que se ha presentado ante el lector es válida.

3.3.2 NFC

NFC es una tecnología inalámbrica, que muchos autores definen como un protocolo wireless en ambos sentidos de muy corto alcance y basado en RFID que permite a un dispositivo leer y/o escribir pequeñas cantidades de datos de otros dispositivos o etiquetas por aproximación.

Esta tecnología inalámbrica trabaja en la banda de los 13,56MHz y es capaz de transmitir a distintas velocidades: 106kbit/s, 212kbit/s o 424kbit/s. Es compatible con otras tecnologías e incluso se puede usar para configurar e iniciar otras conexiones wireless como Bluetooth, WiFi o UltraWireband.

Aprobado como estándar ISO en 2003 (ISO 18092), su uso ya ha tenido recorrido en dispositivos como llaves para el coche, tarjetas de identificación o tickets electrónicos. Aunque su máximo auge está llegando últimamente gracias a la computación y entornos móviles, implantándose esta tecnología en dispositivos móviles como teléfonos.

La principal diferencia del NFC con otras tecnologías inalámbricas como RFID es que el alcance es tan corto que se necesita que los dispositivos a interaccionar estén a escasos centímetros (2-4cm) durante un instante para la transmisión de información. Pese a que esta característica pueda parecer una limitación, es en realidad la clave de esta tecnología.

Al contrario de lo que ocurre con los servicios por radiofrecuencia o bluetooth, basados en el descubrimiento de la presencia del dispositivo en la proximidad; estirar el brazo para acercar un móvil NFC hacia otro dispositivo o etiqueta RFID supone de esta forma apenas un acto reflejo intuitivo y sencillo a la vez que representa en sí una afirmación clara de nuestra voluntad por obtener dichos servicios.

La tecnología NFC al estar basada en tecnologías sin contacto e Identificación por Radio Frecuencia (RFID), necesita de un lector y una etiqueta (o un dispositivo actuando de modo pasivo). El lector puede estar contenido en cualquier dispositivo como un teléfono móvil o bien tratarse de un lector fijo.

Cuando el lector se aproxima a otro lector o a una etiqueta RFID emite una señal de radio de corto alcance que activa el microchip de la etiqueta con lo que podremos leer la pequeña cantidad de datos que se encuentran almacenados en esta. En el caso de la comunicación con etiquetas o tags es el reader o lector el encargado de establecer la comunicación, pero no solamente se permite la transmisión lector NFC-etiqueta, si no que dos dispositivos NFC también pueden comunicarse. En este caso existen varios modos de funcionamiento que se describen a continuación.

En el protocolo NFC siempre hay un dispositivo que inicia la conversación y es este el encargado de monitorizar la misma, rol que se puede intercambiar entre las dos partes implicadas. Existen dos modos de funcionamiento y todos los dispositivos del estándar NFCIP-1 deben soportar ambos modos:

- Modo Activo: Requiere dos dispositivos que generan su propio campo electromagnético para poder transmitir los datos. Ambos dispositivos necesitan energía para funcionar. Este modo es característico de las comunicaciones Peer To Peer (P2P) entre dispositivos NFC.

Figura 9. Ejemplo NFC modo activo

- Modo Pasivo: En este caso, solamente un dispositivo genera el campo electromagnético y el otro se aprovecha de la modulación de la carga para poder

transmitir los datos. El iniciador de la comunicación es el encargado de generar el campo electromagnético.

Figura 10. Ejemplo NFC modo pasivo

3.3.2.1 Elementos básicos y comunicación NFC

Los elementos necesarios para establecer una comunicación vía NFC son:

- Dispositivos NFC
    - Teléfonos móviles
    - Readers o lectores
    - PCs, PDAs, impresoras, electrodomésticos,…
- Etiquetas RFID

En este punto cabe destacar que los dispositivos más habituales y que integran tecnología NFC son teléfonos móviles NFC y Readers NFC. Estos últimos suelen estar conectados a computadores de sobremesa para transmitir y/o recibir los datos de otros dispositivos o etiquetas RFID.

En el apartado anterior se han descrito distintos modos de funcionamiento. Esto ocurre en comunicaciones entre dispositivos NFC y es habitual en verlo en sistemas de pago con esta tecnología, siendo los dispositivos NFC un teléfono móvil y un reader o lector fijo. Al margen de este tipo de comunicación, existen otras variantes, comentadas a continuación.

- *Móvil NFC – Etiqueta RFID:* Otra de las formas de comunicación NFC es la comunicación entre móviles NFC y etiquetas RFID. El dispositivo iniciador, en este caso el móvil y gracias a la energía emitida por el lector contenido en este y al circuito transpondedor integrado en la etiqueta RFID, podemos obtener la información contenida en la etiqueta y actuar en consecuencia. Del mismo modo se podrá escribir en la etiqueta RFID.

- *Reader NFC – Etiqueta RFID:* También es posible la comunicación en la que una etiqueta RFID puede ser leída y/o escrita por un lector o reader fijo. Este reader podrá estar conectado a un PC.

NFC incluye procedimiento de autenticación seguro y mecanismo anticolisiones. Por otra parte toda comunicación NFC consta de 5 fases diferenciadas.

- Descubrimiento
- Autenticación
- Negociación (en la comunicación)
- Transferencia
- Reconocimiento

3.3.2.2 El formato NDEF

El formato NDEF o NFC Data Exchange Format es un formato común registrado por el NFC Forum para poder compartir datos entre los dispositivos NFC y/o entre los dispositivos NFC y las etiquetas. Por lo tanto, las etiquetas que son leídas y/o escritas utilizando este formato, se les conoce como etiquetas NDEF o etiquetas NFC.

NDEF propone una forma de organizar el contenido almacenado en bytes en la etiqueta o que viaja de un dispositivo a otro. Se caracteriza por poseer una cabecera de datos, denominada cabecera NDEF a partir de la cual se encuentran los bloques de información. Cada bloque de información seagrupa en registros que contienen a su vez los datos agrupados en mensajes NDEF y caracterizados por un tipo MIME definido.

Una de las desventajas del formato NDEF es que la totalidad de los bloques de información son accesibles por todos los dispositivos NFC ya que la clave empleada para acceder a estos datos es la clave por defecto (en hexadecimal: FF FF FF FF FF FF). Este hecho propicia que datos escritos previamente en una etiqueta RFID por un dispositivo NFC puedan ser borrados o sobrescritos con otra información en cualquier momento y por cualquier dispositivo. Si esto supusiera un inconveniente, podría ser solventado combinando las ventajas de los formatos NDEF y MIFARE en el que se pone de manifiesto la existencia de claves de acceso.

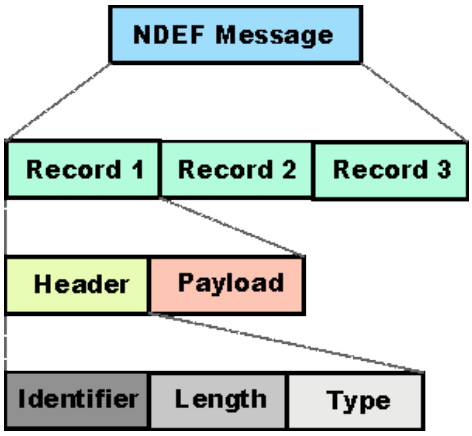


Figura 11. Estructura de mensaje NDEF

Este formato es soportado por la totalidad de dispositivos NFC y constituye un estándar para el intercambio y almacenamiento de información. Por ejemplo, para el desarrollo de aplicaciones NFC en dispositivos móviles NFC usando el estándar NDEF existen un gran número de APIs o librerías que proporcionan los medios necesarios para ello.

3.4 Conclusiones

A lo largo del capítulo se han descrito los fundamentos de tecnología NFC. Teniendo en cuenta sus fundamentos, podemos imaginar una gran variedad de usos prácticos en los cuales dicha tecnología puede estar presente. Hoy en día NFC está en un punto de madurez tecnológica suficiente como para definirse como una tecnología de presente. Sin embargo, la implantación social de NFC a día de hoy es escasa. Varios son los factores que hacen que la presencia de la tecnología NFC en la vida cotidiana de las personas del primer mundo sea limitada o incluso inexistente.

Por un lado la implantación de las capacidades NFC en los dispositivos móviles actuales está siendo lenta. La gran mayoría de los sistemas operativos móviles no da soporte a NFC. Entre éstos se encuentra el iOS, uno de los sistemas operativos para dispositivos móviles de referencia gracias a las diferentes versiones de iPhone e iPad. Los únicos sistemas operativos que actualmente han apostado firmemente por la tecnología NFC son Symbian (en alianza con Nokia, pioneros en cuanto a soporte NFC se refiere) y Android. Este hecho tiene una consecuencia directa, la gran mayoría de los dispositivos móviles existentes en el mundo carecen de esta tecnología, lo que hace que las grandes empresas no apuesten decididamente por su uso masivo. La tendencia natural es que este hecho cambie, y acda vez sean más los dispositivos móviles con NFC, lo que hará que apostar por NFC en un modelo de negocio sea una apuesta segura.

Por otro lado existe un cierto rechazo social al uso de nuevas tecnologías para cuestiones que afectan a la identidad y la seguridad de las personas. Sin embargo los usos de NFC están bastante ligados a dichas cuestiones por su propio fundamento. Por ejemplo se pretende el uso de NFC para control de accesos, pagos, check-ins, etc. Esto supone que la implantación de NFC a nivel social en estas cuestiones debe ser progresiva, hasta que la sociedad pierda el "miedo" a la nueva tecnología presente.

Aun teniendo en cuenta estos factores, parece evidente que NFC acabará desembarcando en la vida de las personas más tarde o más temprano, convirtiéndose en una tecnología de presente con una gran variedad de usos.

Todos estos factores deben ser tenidos muy en cuenta a la hora de abarcar un modelo de negocio en el que se pretenda hacer uso de NFC.

4 Sistemas operativos móviles: Android

4.1 Introducción

Dentro de los teléfonos móviles se encuentra lo que se denomina Smartphone o teléfono inteligente. Estos dispositivos ofrecen características y prestaciones que lo acercan más a un ordenador de sobremesa que a la definición común de móvil.

Entre dichas características se pueden encontrar mejoras en el almacenamiento de datos, mayor capacidad de procesamiento, conexión a internet a través de WI-FI, gran diversidad de software, etc.

Debido a su reducido tamaño, los smartphone quedan todavía lejos de lo que es un ordenador de sobremesa. Sus principales restricciones se encuentran en la velocidad del procesador, el espacio de almacenamiento, el tamaño de su memoria RAM y la autonomía. De todas maneras las tres primeras cada vez se subsanan con mayor rapidez, pudiendo encontrar ya en el mercado, smartphones con procesadores de 4 núcleos a 1,5 GHz, almacenamiento interno de hasta 64 GB y memoria RAM de hasta 2 GB.

El sistema operativo destinado a correr en los móviles debe poseer una gran estabilidad y fiabilidad, ya que incidencias habituales y toleradas en ordenadores de mesa como reinicios y caídas no tienen cabida en estos dispositivos. Además deben amoldarse a las ya consabidas limitaciones de memoria y procesamiento de datos que poseen este tipo de aparatos.

Varios son los sistemas operativos existentes actualmente: iOS, Windows Phone, BlackBerry OS, Android, Symbian, Bada, Meego, etc. De todos ellos podemos destacar por su presunto recorrido en el futuro a tres: iOS, Android y Windows Phone (el último en aparecer, pero con el empuje de la alianza de dos grandes compañías como son Microsoft y Nokia). Todos tienen características comunes y hechos que los diferencian. Por tanto es vital a la hora de elegir el sistema a utilizar en un caso particular, conocer bien sus características para que la elección sea la correcta.

En nuestro caso, usaremos Android para el prototipo, por varias razones que más abajo se detallan, y eso por eso que nos adentraremos más en los entresijos de este sistema operativo.

4.2 Android

Android es una plataforma de software de código abierto que incluye un sistema operativo para dispositivos móviles basado en un kernel Linux (versión 2.6).

El proyecto Android está capitaneado por Google y un conglomerado de otras empresas tecnológicas agrupadas bajo el nombre de Open Handset Alliance. El objetivo principal de esta alianza empresarial (que incluye a fabricantes de dispositivos y operadores, con firmas tan relevantes como Samsung, LG, Telefónica, Intel o Texas Instruments, entre otras muchas) es el desarrollo de estándares abiertos para la telefonía móvil como medida para incentivar su desarrollo y para mejorar la experiencia del usuario.

Con Android se busca reunir en una misma plataforma todos los elementos necesarios que permitan al desarrollador controlar y aprovechar al máximo cualquier funcionalidad ofrecida por un dispositivo móvil (llamadas, mensajes de texto, cámara, agenda de contactos, conexión Wi-Fi, Bluetooth, aplicaciones ofimáticas, videojuegos, etc.), así como poder crear aplicaciones que sean verdaderamente portables, reutilizables y de rápido desarrollo.

4.2.1 Características

Android es una plataforma neutral en cuanto al desarrollo de aplicaciones, es decir, el programador tiene las mismas posibilidades para crear una aplicación que Google o que cualquier empresa perteneciente al Open Handset Alliance. La siguiente lista destaca las características funcionales más importantes de Android:

- Amplia variedad de diseños (VGA, librerías de gráficos 2D y 3D, etc.)
- Almacenamiento de datos en BBDD SQLite.
- Conectividad (GSM/EDGE, CDMA, EV-DO, UMTS, Bluetooth, NFC y Wi-Fi).
- Mensajería (SMS y MMS).

- Navegador Web.
- Máquina virtual de Java.
- Las aplicaciones escritas en Java pueden ser compiladas y ejecutadas en la maquina virtual Dalvik, la cual es una máquina virtual especialmente diseñada para uso en dispositivos móviles.
- Soporte de formatos (MPEG-4, H.264, MP3, AAC, OGG, AMR, JPEG, PNG, GIF).
- Soporte para hardware adicional (cámaras de vídeo, pantallas táctiles, GPS, acelerómetros, etc.).
- Entorno de desarrollo (emulador, herramientas de depuración, perfiles de memoria y funcionamiento, plugin para Eclipse IDE).

Además de las características más puramente funcionales, existen otro tipo de características que es necesario nombrar y hacen de Android un sistema operativo especial.

- Plataforma realmente abierta: Es una plataforma de desarrollo libre basada en Linux y de código libre. Se puede usar y customizar el sistema sin pagar royalties.
- Portabilidad asegurada: Al desarrollar las aplicaciones en Java, y gracias al concepto de maquina virtual, las aplicaciones podrán ser ejecutadas en gran variedad de dispositivos tanto presentes como futuros.
- Arquitectura basada en componentes inspirados en internet: Por ejemplo, las interfaces se hacen en xml, lo que permite el uso de una misma interfaz en dispositivos de pantallas dispares.
- Filosofía de dispositivo siempre conectado a internet.
- Gran cantidad de servicios incorporados: Reconocimiento y síntesis de voz, localización basada en GPS y torres de comunicación, potentes bases de datos, NFC, etc.
- Alto nivel de seguridad: Los programas se encuentran aislados unos de otros. Cada aplicación dispone de una serie de permisos que limitan su rango de actuación.
- Optimización para baja potencia y baja memoria: Por ejemplo el uso de la máquina virtual Dalvik.
- Alta calidad de gráficos y sonido: gran variedad de formatos soportados.

4.2.2 Arquitectura

En las siguientes líneas se dará una visión global por capas de cuál es la arquitectura empleada en Android. Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores.

La capa más inmediata es la corresponde al núcleo de Android. Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluya un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las librerías de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

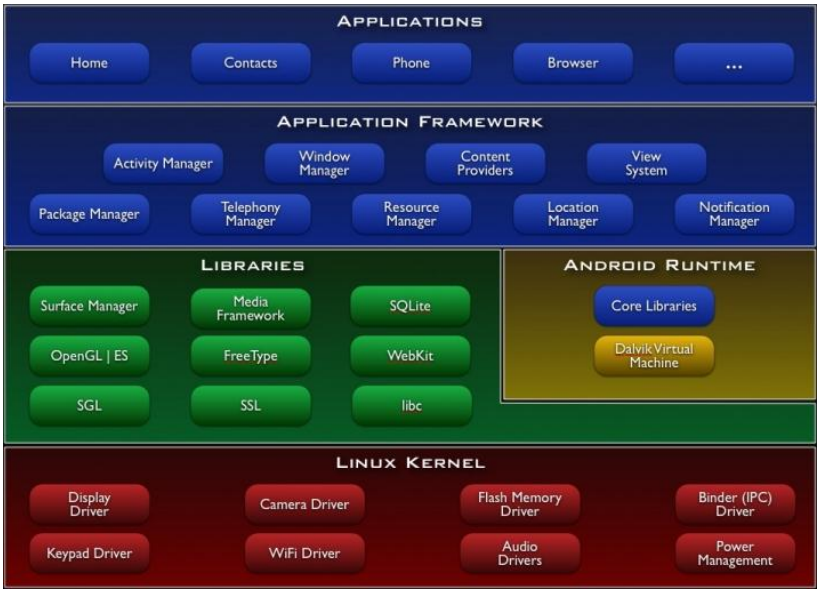


Figura 12. Arquitectura Android

La elección de Linux 2.6 se ha debido principalmente a dos razones: la primera, su naturaleza de código abierto y libre se ajusta al tipo de distribución que se buscaba para Android; la segunda es que este kernel de Linux incluye de por sí numerosos drivers, además de contemplar la gestión de memoria, gestión de procesos, módulos de seguridad, comunicación en red y otras muchas responsabilidades propias de un sistema operativo.

La siguiente capa se corresponde con las librerías utilizadas por Android. Éstas han sido escritas utilizando C/C++ y proporcionan a Android la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de Android.

Entre las librerías más importantes de este nivel, se pueden mencionar las siguientes:

- La librería libc incluye todas las cabeceras y funciones según el estándar del lenguaje C. Todas las demás librerías se definen en este lenguaje.
- La librería Surface Manager es la encargada de componer los diferentes elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.
- OpenGL/SL y SGL representan las librerías gráficas y, por tanto, sustentan la capacidad gráfica de Android. OpenGL/SL maneja gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil, el hardware encargado de proporcionar gráficos 3D. Por otro lado, SGL proporciona gráficos en 2D, por lo que será la librería más habitualmente utilizada por la mayoría de las aplicaciones. Una característica importante de la capacidad gráfica de Android es que es posible desarrollar aplicaciones que combinen gráficos en 3D y 2D.
- La librería Media Libraries proporciona todos los códecs necesarios para el contenido multimedia soportado en Android (vídeo, audio, imágenes estáticas y animadas, etc.)
- FreeType, permite trabajar de forma rápida y sencilla con distintos tipos de fuentes.
- La librería SSL posibilita la utilización de dicho protocolo para establecer comunicaciones seguras.
- A través de la librería SQLite, Android ofrece la creación y gestión de bases de datos relacionales, pudiendo transformar estructuras de datos en objetos fáciles de manejar por las aplicaciones.
- La librería WebKit proporciona un motor para las aplicaciones de tipo navegador, y forma el núcleo del actual navegador incluido por defecto en la plataforma Android.

Al mismo nivel que las librerías de Android se sitúa el entorno de ejecución. Éste lo constituyen las Core Libraries, que son librerías con multitud de clases de Java, y la máquina virtual Dalvik.

Los dos últimos niveles de la arquitectura de Android están escritos enteramente en Java. El framework de aplicaciones representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación.

Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API's y el mismo framework, representado por este nivel.

Entre las API's más importantes ubicadas aquí, se pueden encontrar las siguientes:

- Activity Manager, importante conjunto de la API que gestiona el ciclo de vida de las aplicaciones en Android (del que se hablará más adelante).
- Window Manager, gestiona las ventanas de las aplicaciones y utiliza la librería ya vista Surface Manager.
- Telephone Manager, incluye todas las API's vinculadas a las funcionalidades propias del teléfono (llamadas, mensajes, etc.).
- Content Providers, permite a cualquier aplicación compartir sus datos con las demás aplicaciones de Android. Por ejemplo, gracias a esta API la información de contactos, agenda, mensajes, etc. será accesible para otras aplicaciones.
- View System, proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, mosaicos, botones, check-boxes, tamaño de ventanas, control de las interfaces mediante tacto o teclado, etc. Incluye también algunas vistas estándar para las funcionalidades más frecuentes.
- Location Manager, posibilita a las aplicaciones la obtención de información de localización y posicionamiento, y funcionar según ésta.
- Notification Manager, mediante el cual las aplicaciones, usando un mismo formato, comunican al usuario eventos que ocurran durante su ejecución: una llamada entrante, un mensaje recibido, conexión Wi-Fi disponible, ubicación en un punto determinado, etc. Si llevan asociada alguna acción, en Android

denominada Intent, (por ejemplo, atender una llamada recibida) ésta se activa mediante un simple clic.

- XMPP Service, colección de API's para utilizar este protocolo de intercambio de mensajes basado en XML.

El último nivel del diseño arquitectónico de Android son las aplicaciones. Éste nivel incluye tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo. Todas estas aplicaciones utilizan los servicios, las API's y librerías de los niveles anteriores.

4.2.3 La máquina virtual Dalvik

En Android, todas las aplicaciones se programan en el lenguaje Java y se ejecutan mediante una máquina virtual de nombre Dalvik, específicamente diseñada para Android. Esta máquina virtual ha sido optimizada y adaptada a las peculiaridades propias de los dispositivos móviles (menor capacidad de proceso, baja memoria, alimentación por batería, etc.) y trabaja con ficheros de extensión .dex (Dalvik Executables). Dalvik no trabaja directamente con el bytecode de Java, sino que lo transforma en un código más eficiente que el original, pensado para procesadores pequeños.

Gracias a la herramienta "dx", esta transformación es posible: los ficheros .class de Java se compilan en ficheros .dex, de forma que cada fichero .dex puede contener varias clases. Después, este resultado se comprime en un único archivo de extensión .apk (Android Package), que es el que se distribuirá en el dispositivo móvil. Dalvik permite varias instancias simultáneas de la máquina virtual, y a diferencia de otras máquinas virtuales, está basada en registros y no en pila, lo que implica que las instrucciones son más reducidas y el número de accesos a memoria es menor. Así mismo, Dalvik no permite la compilación Just-in-Time.

Según los responsables del proyecto, la utilización de esta máquina virtual responde a un deseo de mejorar y optimizar la ejecución de aplicaciones en dispositivos móviles, así como evitar la fragmentación de otras plataformas como Java ME. A pesar de esta aclaración oficial, no ha pasado inadvertido que con esta maniobra Android ha esquivado tener que utilizar directamente Java ME, evitando así que los futuros desarrolladores de

aplicaciones tengan que comprar ninguna licencia, ni tampoco que publicar el código fuente de sus productos. El lenguaje Java utilizado en Android no sigue ningún JSR (Java Specification Requests) determinado, y Google se afana en recordar que Android no es tecnología Java, sino que simplemente utiliza este lenguaje en sus aplicaciones.

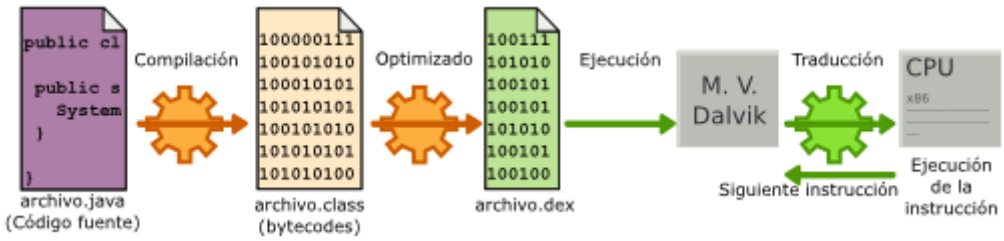


Figura 13. Proceso de transformación y ejecución de código Android

Todas las aplicaciones en Android pueden descomponerse en cuatro tipos de bloques o componentes principales. Cada aplicación será una combinación de uno o más de estos componentes, que deberán ser declarados de forma explícita en un fichero con formato XML denominado "AndroidManifest.xml", junto a otros datos asociados como valores globales, clases que implementa, datos que puede manejar, permisos, etc. Este fichero es básico en cualquier aplicación en Android y permite al sistema desplegar y ejecutar correctamente la aplicación.

A continuación se exponen los cuatro tipos de componentes en los que puede dividirse una aplicación para Android.

4.2.3.1 Activity

Sin duda es el componente más habitual e indispensable de las aplicaciones para Android. Un componente Activity refleja una determinada actividad llevada a cabo por una aplicación, y que lleva asociada típicamente una ventana o interfaz de usuario. Es importante señalar que no contempla únicamente el aspecto gráfico, sino que éste forma parte del componente Activity a través de vistas representadas por clases como View y sus derivadas. Este componente se implementa mediante la clase de mismo nombre Activity.

La mayoría de las aplicaciones permiten la ejecución de varias acciones a través de la existencia de una o más pantallas. Por ejemplo, piénsese en una aplicación de mensajes de

texto. En ella, la lista de contactos se muestra en una ventana. Mediante el despliegue de una segunda ventana, el usuario puede escribir el mensaje al contacto elegido, y en otra tercera puede repasar su historial de mensajes enviados o recibidos. Cada una de estas ventanas debería estar representada a través de un componente Activity, de forma que navegar de una ventana a otra implica lanzar una actividad o dormir otra. Android permite controlar por completo el ciclo de vida de los componentes Activity.

Muy vinculado a este componente se encuentran los Intent, una interesante novedad introducida por Android.

4.2.3.2 Intent

Un Intent consiste básicamente en la voluntad de realizar alguna acción, generalmente asociada a unos datos. Lanzando un Intent, una aplicación puede delegar el trabajo en otra, de forma que el sistema se encarga de buscar qué aplicación entre las instaladas es la que puede llevar a cabo la acción solicitada. Por ejemplo, abrir una URL en algún navegador web, o escribir un correo electrónico desde algún cliente de correo.

4.2.3.3 Broadcast Intent Receiver

Un componente Broadcast Intent Receiver se utiliza para lanzar alguna ejecución dentro de la aplicación actual cuando un determinado evento se produzca (generalmente, abrir un componente Activity).

Por ejemplo, una llamada entrante o un SMS recibido. No tiene interfaz de usuario asociada, pero puede utilizar la API de Notification Manager, mencionada anteriormente, para avisar al usuario del evento producido a través de la barra de estado del dispositivo móvil. Este componente se implementa a través de una clase de nombre BroadcastReceiver.

Para que Broadcast Receiver funcione, no es necesario que la aplicación en cuestión sea la aplicación activa en el momento de producirse el evento. El sistema lanzará la aplicación si es necesario cuando el evento monitorizado tenga lugar.

4.2.3.4 Service

Un componente Service representa una aplicación ejecutada sin interfaz de usuario, y que generalmente tiene lugar en segundo plano mientras otras aplicaciones (éstas con interfaz) son las que están activas en la pantalla del dispositivo.

Un ejemplo típico de este componente es un reproductor de música. La interfaz del reproductor muestra al usuario las distintas canciones disponibles, así como los típicos botones de reproducción, pausa, volumen, etc. En el momento en el que el usuario reproduce una canción, ésta se escucha mientras se siguen visionando todas las acciones anteriores, e incluso puede ejecutar una aplicación distinta sin que la música deje de sonar. La interfaz de usuario del reproductor sería un componente Activity, pero la música en reproducción sería un componente Service, porque se ejecuta en background. Este elemento está implementado por la clase de mismo nombre Service.

4.2.3.5 Content Provider

Con el componente Content Provider, cualquier aplicación en Android puede almacenar datos en un fichero, en una base de datos SQLite o en cualquier otro formato que considere.

Además, estos datos pueden ser compartidos entre distintas aplicaciones. Una clase que implemente el componente Content Provider contendrá una serie de métodos que permite almacenar, recuperar, actualizar y compartir los datos de una aplicación.

Existe una colección de clases para distintos tipos de gestión de datos en el paquete android.provider. Además, cualquier formato adicional que se quiera implementar deberá pertenecer a este paquete y seguir sus estándares de funcionamiento.

4.2.4 Ciclo de vida de una aplicación Android

En Android, cada aplicación se ejecuta en su propio proceso. Esto aporta beneficios en cuestiones básicas como seguridad, gestión de memoria, o la ocupación de la CPU del dispositivo móvil. Android se ocupa de lanzar y parar todos estos procesos, gestionar su

ejecución y decidir qué hacer en función de los recursos disponibles y de las órdenes dadas por el usuario.

El usuario desconoce este comportamiento de Android. Simplemente es consciente de que mediante un simple clic pasa de una a otra aplicación y puede volver a cualquiera de ellas en el momento que lo desee. No debe preocuparse sobre cuál es la aplicación que realmente está activa, cuánta memoria está consumiendo, ni si existen o no recursos suficientes para abrir una aplicación adicional. Todo eso son tareas propias del sistema operativo.

Android lanza tantos procesos como permitan los recursos del dispositivo. Cada proceso, correspondiente a una aplicación, estará formado por una o varias actividades independientes (componentes Activity) de esa aplicación. Cuando el usuario navega de una actividad a otra, o abre una nueva aplicación, el sistema duerme dicho proceso y realiza una copia de su estado para poder recuperarlo más tarde. El proceso y la actividad siguen existiendo en el sistema, pero están dormidos y su estado ha sido guardado. Es entonces cuando crea, o despierta si ya existe, el proceso para la aplicación que debe ser lanzada, asumiendo que existan recursos para ello.

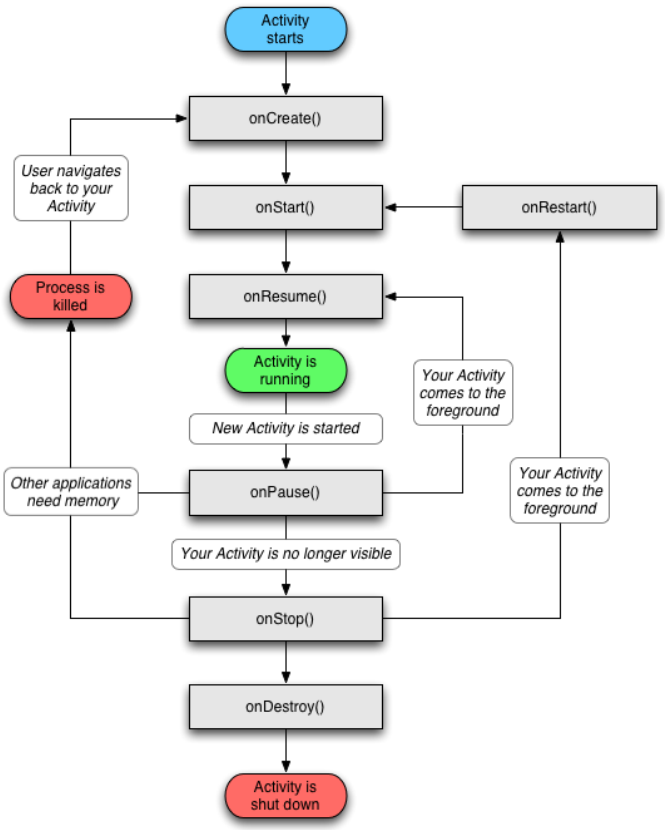


Figura 14. Ciclo de vida de un elemento Activity

Cada uno de los componentes básicos de Android tiene un ciclo de vida bien definido; esto implica que el desarrollador puede controlar en cada momento en qué estado se encuentra dicho componente, pudiendo así programar las acciones que mejor convengan. El componente Activity, probablemente el más importante, tiene un ciclo de vida como el mostrado en la figura anterior.

De la Figura pueden sacarse las siguientes conclusiones:

- onCreate(), onDestroy(): abarcan todo el ciclo de vida. Cada uno de estos métodos representan el principio y el fin de la actividad.
- onStart(), onStop(): representan la parte visible del ciclo de vida. Desde onStart() hasta onStop(), la actividad será visible para el usuario, aunque es posible que no tenga el foco de acción por existir otras actividades superpuestas con las que el usuario está interactuando. Pueden ser llamados múltiples veces.
- onResume(), onPause(): delimitan la parte útil del ciclo de vida. Desde onResume() hasta onPause(), la actividad no sólo es visible, sino que además tiene el foco de la acción y el usuario puede interactuar con ella.

Tal y como se ve en el diagrama de la Figura, el proceso que mantiene a esta Activity puede ser eliminado cuando se encuentra en onPause() o en onStop(), es decir, cuando no tiene el foco de la aplicación. Android nunca elimina procesos con los que el usuario está interactuando en ese momento. Una vez se elimina el proceso, el usuario desconoce dicha situación y puede incluso volver atrás y querer usarlo de nuevo. Entonces el proceso se restaura gracias a una copia y vuelve a estar activo como si no hubiera sido eliminado. Además, la Activity puede haber estado en segundo plano, invisible, y entonces es despertada pasando por el estado onRestart().

Pero, ¿qué ocurre en realidad cuando no existen recursos suficientes? Obviamente, los recursos son siempre limitados, más aun cuando se está hablando de dispositivos móviles. En el momento en el que Android detecta que no hay los recursos necesarios para poder lanzar una nueva aplicación, analiza los procesos existentes en ese momento y elimina los procesos que sean menos prioritarios para poder liberar sus recursos.

Cuando el usuario regresa a una actividad que está dormida, el sistema simplemente la despierta. En este caso, no es necesario recuperar el estado guardado porque el proceso todavía existe y mantiene el mismo estado. Sin embargo, cuando el usuario quiere regresar a una aplicación cuyo proceso ya no existe porque se necesitaba liberar sus recursos, Android lo crea de nuevo y utiliza el estado previamente guardado para poder restaurar una copia nueva del mismo. Como se ya ha explicado, el usuario no percibe esta situación ni conoce si el proceso ha sido eliminado o está dormido.

4.2.5 Política de eliminación de procesos

Tal y como se explica en el apartador anterior, cada aplicación Android se ejecuta en su propio proceso. Este proceso se crea cada vez que una aplicación necesita ejecutar parte de su código, y seguirá existiendo hasta que la aplicación finalice o hasta que el sistema necesite utilizar parte de sus recursos para otra aplicación considerada prioritaria.

Por ello, es importante saber cómo los componentes de una aplicación en Android (Activity, Broadcast Intent Receiver, Service y Content Provider) determinan e influyen en el ciclo de vida de la aplicación. No usar los componentes correctamente a la hora de construir una aplicación puede significar que el sistema operativo la termine cuando en realidad está haciendo una tarea importante para el usuario.

Android construye una jerarquía donde evalúa la clase de componentes que están ejecutándose y el estado de los mismos. En orden de importancia, serían los siguientes:

1. Procesos en primer plano: aquellos necesarios para lo que el usuario está haciendo en ese momento. Un proceso se encuadra en esa categoría si cumple alguna de las siguientes condiciones:

a. Tiene un componente Activity ejecutándose, con la que el usuario está interactuando.

b. Tiene un componente Broadcast Intent Receiver ejecutándose.

c. Ha lanzado algún otro proceso que tiene un componente Service ejecutándose en el momento.

Idealmente, sólo debería haber algunos de estos procesos en el sistema, y su eliminación debería darse únicamente en casos extremos en los que la falta de recursos impide que sigan ejecutándolos todos.

2. Procesos visibles: aquellos procesos que contienen un componente Activity visible en la pantalla, pero no con el foco de actividad en ese momento.

3. Procesos de servicio: aquellos procesos que tienen un componente Service y que están ejecutándose en background. Aunque no sean visibles directamente al usuario, desempeñan tareas sí percibidas por este.

4. Procesos en segundo plano: procesos con un componente Activity, que no son visibles al usuario. Estos procesos no tienen una importancia directa para el usuario en ese momento.

5. Procesos vacíos: procesos que ya no ejecutan ninguna actividad, pero que se mantienen en memoria para agilizar una posible nueva llamada por parte del usuario.

Según esta jerarquía, Android prioriza los procesos existentes en el sistema y decide cuáles han de ser eliminados, con el fin de liberar recursos y poder lanzar la aplicación requerida.

4.2.6 Seguridad en Android

En Android, como ya se ha comentado, cada aplicación se ejecuta en su propio proceso. La mayoría de las medidas de seguridad entre el sistema y las aplicaciones deriva de los estándares de Linux 2.6, cuyo kernel, recuérdese, constituye el núcleo principal de Android. Cada proceso en Android constituye lo que se llama un cajón de arena o sandbox, que proporciona un entorno seguro de ejecución.

Por defecto, ninguna aplicación tiene permiso para realizar ninguna operación o comportamiento que pueda impactar negativamente en la ejecución de otras aplicaciones o del sistema mismo. Por ejemplo, acciones como leer o escribir ficheros privados del usuario (contactos, teléfonos, etc.), leer o escribir ficheros de otras aplicaciones, acceso de red, habilitación de algún recurso hardware del dispositivo, etc., no están permitidas. La única forma de poder saltar estas restricciones impuestas por

Android, es mediante la declaración explícita de un permiso que autorice a llevar a cabo una determinada acción habitualmente prohibida.

Además, en Android es obligatorio que cada aplicación esté firmada digitalmente mediante un certificado, cuya clave privada sea la del desarrollador de dicha aplicación. No es necesario vincular a una autoridad de certificado, el único cometido del certificado es crear una relación de confianza entre las aplicaciones. Mediante la firma, la aplicación lleva adjunta su autoría.

Para establecer un permiso para una aplicación, es necesario declarar en el manifiesto (archivo AndroidManifest.xml) uno o más elementos <uses-permission> donde se especifica el tipo de permiso que se desea habilitar.

En la clase android.Manifest.permission se especifican todos los posibles permisos que se pueden conceder a una aplicación: utilización de Wi-Fi, Bluetooth, llamadas telefónicas, cámara, Internet, mensajes SMS y MMS, vibrador, etc

El elemento <uses-permission> contempla una serie de atributos que definen y matizan el alcance del permiso dado:

- android:name: especificación del permiso que se pretende conceder. Debe ser un nombre de alguno de los listados en la clase android.Manifest.permission.
- android:label: una etiqueta o nombre convencional fácilmente legible para el usuario.
- android:permissionGroup: permite especificar un grupo asociado al permiso. Los posibles grupos se encuentran listados en la clase android.Manifest.permission_group y pueden tener valores como ACCOUNTS (cuentas válidas de Google), COST_MONEY (acciones que llevan vinculadas un pago) o PHONE_CALLS (acciones relacionadas con llamadas), entre otros.
- android:protectionLevel: determina el nivel de riesgo del permiso, y en función del mismo influye en cómo el sistema otorga o no el permiso a la aplicación. Oscila entre valores desde el 0 hasta el 3.
- android:description: descripción textual del permiso.
- android:icon: icono gráfico que puede ser asociado al permiso

4.3 Sistemas operativos móviles y NFC

4.3.1.1 Introducción

Los sistemas de comunicación NFC (Near Field Communication) representan nuevas posibilidades funcionales a través de los dispositivos móviles actuales, especialmente en smartphones. Cada día son más los fabricantes de dispositivos móviles que incorporan esta tecnología en sus terminales. El hecho de integrar a nivel de hardware la teconología NFC en los terminales no es quizá el mayor inconveniente que tiene NFC para su popularización.

Probablemente una de las causas de su lenta incorporación a la vida cotidiana de los usuarios (en España prácticamente nula incorporación actualmente), se debe principalmente a la capacidad de las versiones de los sistemas operativos móviles de proveer interfaces de programación que puedan explotar toda la potencialidad de esta tecnología por los desarrolladores de software a través de sus APIs.

Ultimamente, la dirección tomada por las empresas creadoras de los sistemas operativos es la de proveer a través de sus APIs de esta funcionalidad. Por ejemplo, podemos ver como Android, especialmente desde su versión 4 (Ice Cream Sandwich), o BlackBerry OS 7 permiten a los desarrolladores disponer de las herramientas necesarias para poder crear aplicaciones de valor añadido a través de la tecnología NFC incorporada en sus teléfonos.

En los apartados siguientes analizaremos qué sistemas operativos dan soporte a esta tecnología y que estrategia de futuro tienen otros sistemas operativos que todavía no permiten el uso de NFC de cara al futuro.

4.3.2 Android

Como ya hemos avanzado, sin el soporte de los sistemas operativos móviles a la tecnología NFC y la capacidad por parte del desarrollador de hacer uso de ella para la creación de nuevas aplicaciones, sería imposible la implantación social de NFC. Son los sistemas operativos,

por tanto, uno de los principales actores en el escenario presente de la integración de nuevas tecnologías en la vida cotidiana de las personas.

En el caso de Android, desde la aparición de la versión 2.3 o Gingerbread (nivel de API 9) en diciembre de 2010, hace poco más de año y medio, este sistema operativo provee de funcionalidades NFC a través de su API. El inconveniente es que los teléfonos coetáneas a la salida de este SO no tenían los chips NFC en sus terminales, con lo que la adaptación de hardware a las nuevas posibilidades de Android se produce de forma lenta conforme los usuarios van renovando sus terminales. Además uno de los grandes inconvenientes de la API 9 de Android respecto a NFC es que solo soporta la funcionalidad de lectura. Lo cual es algo que limita en exceso las posibilidades de NFC, haciendo que su uso sea residual entre las aplicaciones existentes. La falta de terminales capaces y la limitación de las propias capacidades de Android hicieron que el hecho del soporte NFC se quedara más en el nivel teórico que en el práctico.

Pero, la apuesta de Android por dar soporte completo a NFC era más que decidida. Así pues, en la revisión propuesta por para Android 2.3, la 2.3.3 (nivel de API 10) se introdujo el soporte completo a NFC, en cuanto que ya era posible tanto la lectura como la escritura y la transmisión.

Con la transmisión se puede establecer comunicación entre dispositivos móviles e intercambiar información, esto es, incluye conexión peer-to-peer. Algo absolutamente necesario para poder sacar toda la funcionalidad que esta tecnología puede proveer. La escritura permitirá escribir etiquetas con nueva información. Abriendo todavía más a nuevas posibilidades incluso en ámbitos profesionales especializados como la logística, etc.

El principal problema de la API 10 de Android era que había salido en febrero de 2011, junto a la API 11 especializada para tablets, y tuvo una revisión en Julio del mismo año, cuando Google ya había anunciado a los fabricantes que durante ese mismo año saldría la nueva versión del sistema operativo (Android 4, Ice Cream Sandwich), y por tanto que la API 10 no tendría mucho recorrido. Por ende los fabricantes no se centraron tanto en estas nuevas funcionalidades y esperaron al enfoque que Google dió a Android en la nueva versión 4 que fusiona el sistema operativo de smartphones y tablets, y que es sin duda la gran apuesta de presente de Android, siendo mucho más atractivo al usuario y potente que sus antecesores Gingerbread y Honeycomb.

El Ice Cream Sandwich también trae consigo algunas novedades en cuanto a NFC se refiere, a través de su aplicación Android Beam. Ya no solo se puede utilizar NFC, sino que ahora hay disponible una API dedicada a la transferencia de información entre terminales vía NFC. Y lo mejor es que el sistema funciona tipo Push, por lo que el receptor del mensaje no tendrá que iniciar la aplicación de destino para recibir el mensaje, sino que Android Beam lanzará un Intent cuando esté el mensaje disponible. Algo que facilita sobremanera el uso transparente de esta nueva tecnología.

4.3.3 Otros sistemas operativos y NFC

Veamos como afrontan los demás sistemas operativos, con cierta presencia en el mercado, el reto de integrar NFC.

4.3.3.1 Windows Phone 7.5

Aunque el desarrollador y product manager de Windows Will Coleman aseguro a finales de 2011 que Windows Phone daría soporte a NFC, a los pocos días fue desmentido por parte de la compañía. Por tanto WP no tiene ningun API en este sentido y es imposible poder dar funcionalidades a los teléfonos que montan este sistema operativo en la actualidad.

Aunque uno de los principales fabricantes que ha apostado por Windows Phone, Nokia, ya tenga experiencia en la integración de NFC en sus terminales en la época de Symbian, al no estar disponible en el sistema operativo de sus nuevos terminales, no podrán hacer uso de NFC. Aún así, Nokia acaba de lanzar algunos terminales con Windows Phone que sí tienen el chip NFC necesario, a la espera de que con la actualización a Windows Phone 8 en un futuro el sistema operativo le de soporte.

4.3.3.2 BlackBerry 7

A principios de Abril de 2011, solo un poco más tarde que Android, BlackBerry daba soporte NFC a sus samrtphones a través de su nuevo sistema operativo BlackBerry OS 7, que en ese momento solo podía estar disponible en su modelo Bold 9900. Aunque muchos dan por muerto al producto estrella de RIM, la empresa ha demostrado estar a la altura de las

circunstancias en cuanto a incorporar novedades tecnológicas que otras empresas de mucho mayor peso no han podido todavía.

El nuevo sistema operativo dispone de API para desarrolladores en cuanto a NFC se refiera, como no podía ser de otra manera, además de funcionalidades NFC incorporadas por defecto en su SO. Conforme salen nuevos modelos de BB al mercado con BB OS 7, se van ampliando la gama de BBs con funcionalidad NFC entre sus usuarios.

4.3.3.3 iOS 5

El nuevo sistema opertivo de los de Cupertino, el iOS 5, no trajo ninguna novedad en cuanto a NFC se refiere. iPhone y NFC nunca han sido sinónimos de momento, pero la compañía tendrá que ceder en este sentido y acomodarse dentro de las nuevas circunstancias.

Es casi un secreto a voces que el iPhone 5 contará con conectividad NFC y parece que desde Cupertino quieren utilizar esta nueva posibilidad de conexión inalámbrica para algo más que el pago desde el móvil.

Las últimas patentes que está registrando Apple en torno a la tecnología NFC dan pistas de que la compañía de la manzana quiere exprimir esta nueva conexión para aportar otras funcionalidades al iPhone 5, nada nuevo en otros sistemas operativos. Pero, eso sí, seguramente aprovecharán su fuerte integración y su gran presencia en el mercado para revolucionar hasta el último centímetro de la Tierra cuando presenten estas funcionalidades. Quizá este hecho también suponga el despegue necesario y definitivo de NFC, que tantas posibilidades alberga.

4.4 Conclusión

Existe hoy en día una gran oferta de sistemas operativos para dispositivos móviles. A la hora de desarrollar un nuevo proyecto, es necesario conocer las características principales, usuarios potenciales y perspectivas de futuro de cada uno de ellos, con la intención de hacer la elección correcta.

Para el prototipo que se desprende del proyecto se decide utilizar el sistema operativo Android por varias razones. En primer lugar, Android es open source, y ello implica entre otras

muchas cosas que los costes de desarrollo de aplicaciones que para ser probadas necesitan ser ejecutadas en un dispositivo físico, puesto que el emulador no puede emular todas las funciones, su coste se prácticamente nulo. Este hecho ya eliminaría de la lista de posibles candidatos a sistemas operativos tan punteros como iOS o Windows Phone. Además Android ofrece un gran soporte a NFC, algo difícil de ver en los demás sistemas, por lo que sin lugar a dudas es el sistema operativo correcto para lo que aquí se pretende.

5 SOA y Servicios Web

5.1 Introducción

Los sistemas informáticos tradicionales se han organizado en grandes bloques monolíticos que contienen tanto los procesos de negocio como sus funciones automatizadas. Estos sistemas han conseguido una gran mejora de productividad en las empresas, automatizando procesos de negocio, pero su concepción monolítica hace que los cambios y adaptaciones a las nuevas necesidades tiendan a ser más lentos y costosos de lo deseable. En bastantes organizaciones esto provoca que los sistemas marchen por detrás de las necesidades de negocio.

Para conseguir un mayor nivel de agilidad es necesario poder combinar rápidamente los distintos componentes del sistema, algo a lo que la concepción monolítica tradicional plantea muchas restricciones. La arquitectura SOA separa los procesos de negocio de las funciones automatizadas y organiza estas últimas en módulos individuales catalogados en un diccionario de servicios que permiten su utilización por parte de toda la organización.

SOA no es solamente una tecnología, sino una arquitectura que trata de estructurar las aplicaciones de negocio y la tecnología para responder de forma ágil y flexible a las demandas del mercado. La importancia de la arquitectura SOA, y probablemente la razón por la que despierta tanto interés es que ofrece una oportunidad real de situar las tecnologías de la información en un nuevo nivel, convirtiéndolas en auténticos habilitadores del negocio. Los servicios web son la forma más común de implementar SOA. Podemos distinguir dos grandes tipos, los basados en SOAP y los basados en REST.

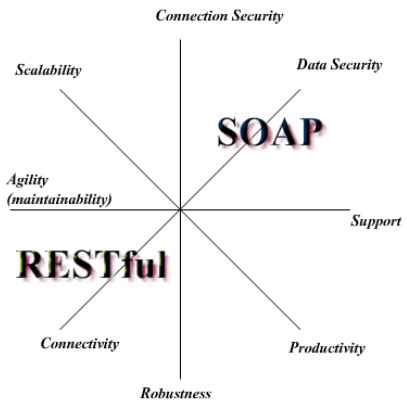


Figura 15. SOAP vs REST según varios criterios

5.2 Arquitecturas Orientadas a Servicios (SOA)

5.2.1 Introducción

Las arquitecturas orientadas a servicios (SOA) son arquitecturas de software que definen el uso de servicios como soporte a los requisitos del negocio, cuyo objetivo es alcanzar el mínimo acoplamiento posible entre agentes software. Un servicio es una unidad de trabajo realizada por un proveedor de servicios para alcanzar un resultado final deseado por un consumidor del servicio. Tanto el proveedor como el consumidor del servicio son roles realizados por agentes software en lugar de sus propietarios.

En un sentido general, una arquitectura orientada a servicios es una solución software que pretende permitir a la empresa organizar y hacer uso de múltiples procesos. Con SOA, las aplicaciones software ya no son enormes bloques de funciones y procesos. En cambios, estas aplicaciones se componen de servicios modulares ensamblados. Recordemos que un servicio es una función software simple (como por ejemplo, cancelar la reproducción de un CD). Puede ser ejecutada bajo demanda por cualquier sistema, sin tener en cuenta el sistema operativo, plataforma, lenguaje de programación o posición geográfica.

Lo que es revolucionario acerca de SOA no es el concepto en sí mismo, el cual ha estado presente desde hace tiempo, sino el hecho de que se puede implementar a través de la WWW (World Wide Web). De la misma forma que las páginas web se cargan en cualquier plataforma, los servicios web trabajan de forma similar, sin tener en cuenta la plataforma, ya que se construyen utilizando estándares universales.

La idea de SOA parte directamente de la programación orientada a objetos, la cual sugiere una relación entre los datos y su procesamiento. Así, definen los servicios formalmente a través de interfaces independientes de la plataforma subyacente y del lenguaje de programación. Estas interfaces ocultan las particularidades de su implementación, lo que las hace independientes del desarrollador y del lenguaje de programación. A través de estas arquitecturas se consiguen diseñar e implementar sistemas altamente escalables que reflejan la lógica de negocio y a la vez facilitan la interacción entre distintos sistemas propietarios o de terceros. La razón por la cual queremos que alguien haga el trabajo por nosotros es porque es

experto. Utilizar un servicio es generalmente más barato y efectivo que hacerlo por nosotros mismos. Así, la mayoría de nosotros comprendemos que no podemos ser expertos en todo. La misma regla se puede aplicar a la construcción de sistemas software.

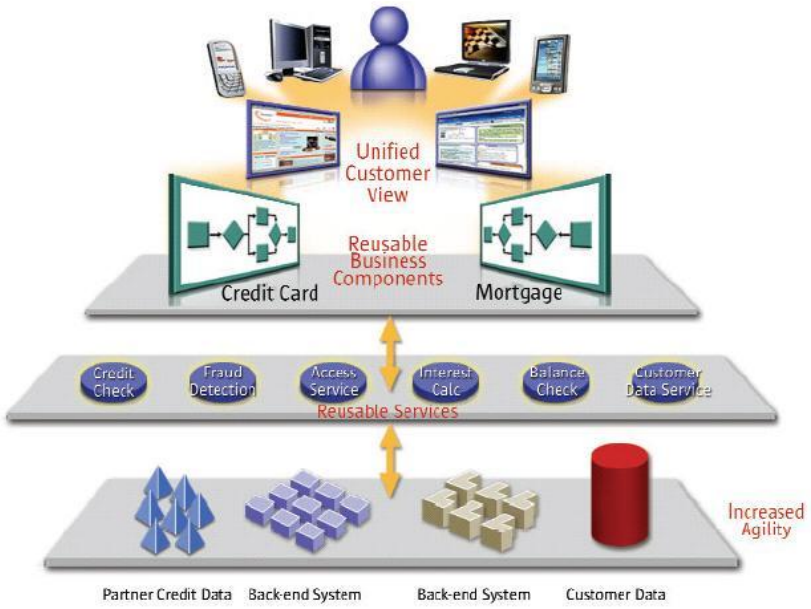


Figura 16. Arquitectura orientada a servicios

5.2.2 Principios de diseño

Para conseguir desacoplar los agentes software que interactúan en una arquitectura SOA han de utilizarse dos principios básicos inherentes a la propia definición de la arquitecura:

1. Escoger un pequeño conjunto de interfaces simples y distribuidas para todos los agentes software participantes. Sólo la semántica genérica es codificada en las interfaces. Éstas deberían estar disponibles universalmente para todos los proveedores y consumidores.

2. Definir mensajes descriptivos por un esquema extensible que se entrega a través de las interfaces. No se ofrece información sobre el funcionamiento del sistema a través de mensajes. Los esquemas son los encargados de limitar el vocabulario y estructura de los mensajes.

Las interfaces son tremendamente importantes, si éstas no han sido bien definidas o no funcionan, el sistema no funciona. Integrar más interfaces es caro, y además incrementa la posibilidad de errores en aplicaciones distribuidas. Las interfaces remotas son la parte más lenta de la mayoría de las aplicaciones distribuidas. Con todo esto, en lugar de construir nuevas interfaces para cada aplicación, tiene más sentido reutilizar unas genéricas para todas las aplicaciones. Ello implica el uso de servicios de grano grueso.

Así, ya que sólo disponemos de unas pocas interfaces genéricas disponibles, debemos incluir dentro de los mensajes semántica específica sobre las aplicaciones. Podemos enviar cualquier tipo de mensaje sobre nuestras interfaces, pero hay unas pocas reglas a seguir para poder decir que una arquitectura es orientada a servicio.

- Mensajes descriptivos: Los mensajes deben ser descriptivos, más que instructivos, porque el proveedor del servicio es responsable de resolver el problema. Por ejemplo, sería similar a la situación de ir a un restaurante y decir al camarero lo que te gustaría tomar y tus preferencias, pero no deberíamos explicar al cocinero cómo cocinar tus platos, paso por paso.

- Estandarización de mensajes (contrato): los proveedores de servicio no serán capaces de entender tu petición si tus mensajes no están escritos en un formato, estructura y vocabulario comprensible por todos los involucrados. Así, limitar el vocabulario y la estructura de los mensajes es una necesidad para lograr una comunicación eficiente. Cuanto más restringido sea el mensaje, más sencillo es entenderlo.

- Extensibilidad: La posibilidad de realizar extensiones es de vital importancia. El mundo es un sitio cambiante en todo momento, y de forma similar es el entorno en el que vive el software. Estos cambios demandan cambios correspondientes en el sistema software, consumidores de servicios, proveedores, y los mensajes que intercambian. Si los mensajes no son extensibles, los consumidores y proveedores estarán bloqueados en una versión concreta del servicio. Restricción y extensibilidad están profundamente relacionadas, se necesitan ambas, e incrementar una supone una reducción de la otra. Lo ideal es lograr un correcto balance.

- Descubrimiento: SOA debe poseer un mecanismo que permita al consumidor descubrir un proveedor de servicios bajo el contexto de un servicio buscado por el consumidor. El mecanismo debe ser flexible, y no debería ser un registro centralizado.

Existen además un número de posibilidades adicionales que se pueden aplicar en SOA para mejorar su escalabilidad, rendimiento y fiabilidad:

- Servicio stateless (sin estados): Cada mensaje que envía un consumidor a un proveedor debe contener toda la información necesaria para que el proveedor la procese. Esta restricción hace más escalable a un proveedor de servicio puesto que el proveedor no tiene que almacenar información de estado entre peticiones. Esto se puede denominar "servicio de producción en masa" ya que cada petición puede ser tratada de manera genérica. Esta restricción además provee de más visibilidad, ya que cualquier software de monitorización puede inspeccionar una petición independiente y extraer su intención. Esto permite además la recuperación de forma sencilla de fallos parciales, puesto que no hay estados intermedios, haciendo el servicio más fiable.

- Servicio stateful (con estados): Los servicios con estados son necesarios en diversas situaciones. Por ejemplo, al establecer una sesión entre el consumidor y el proveedor. Las sesiones se establecen típicamente por razones de eficiencia. Enviar un certificado de seguridad en cada petición es una carga importante, tanto para el consumidor como para el proveedor, es mucho más rápido remplazar el certificado con un token compartido entre ambos. Otra situación es la de ofrecer un servicio personalizado. Estos servicios requieren que tanto el consumidor con el proveedor compartan el mismo contexto, incluido en o referenciado mediante mensajes intercambiados entre el consumidor y proveedor. El inconveniente de este requisito es que puede reducir la escalabilidad del proveedor de servicio, ya que puede necesitar recordar el contexto para cada consumidor. Además, incrementa el acoplamiento entre el proveedor de servicio y el consumidor, y hace que el cambio entre proveedores sea más difícil. No se aconseja salvo estricta necesidad.

- Peticiones idempotentes (que no realizan ningún cambio): Las peticiones duplicadas recibidas por un agente software tienen el mismo efecto que una petición única. Este requisito permite que los proveedores y consumidores incrementen la fiabilidad total del servicio, simplemente repitiendo la petición si ha habido algún fallo.

5.2.3 Conclusiones

Podemos decir entonces, que SOA no es una herramienta, sino un conjunto de patrones de construcción de nuevas aplicaciones más dinámicas y menos dependientes. SOA facilita el acceso a la lógica de negocios y la información entre diversos servicios de una manera sistemática, pudiendo además orquestar diversos servicios remotos para componer uno único. La mayoría de las definiciones identifican la utilización de Servicios Web, los cuales veremos en el siguiente apartado con más profundidad, en la implementación de una arquitectura orientada a servicios. No obstante, se puede implementar utilizando cualquier otra tecnología basada en servicios.

5.3 Servicios Web

5.3.1 Introducción

Los Servicios Web se han convertido en la implementación más utilizada en arquitecturas orientadas a servicios. Esto se debe a que poseen un conjunto de características que permiten cubrir todos los principios básicos de la orientación a servicios.

El concepto de Servicio Web puede resultar confuso, ya que no existe una definición única universalmente aceptada sobre lo que son y lo que el concepto engloba. Los Servicios Web surgieron como un conjunto de protocolos, estándares y recomendaciones, definidos por la W3C (World Wide Web Consistorium) y OASIS (Organization for the Advancement of Structured Information Standards), para lograr la interoperabilidad en la interacción entre máquinas, sistemas software y aplicaciones a través de la red. La definición de Servicio Web ha estado siempre bajo debate en el grupo de trabajo de la arquitectura de Servicios Web del W3C.

Se acepta generalmente que un Servicio Web es una SOA con restricciones adicionales:

1. Las interfaces se deben basar en protocolos de Internet como HTTP, FTP y SMTP.
2. Los mensajes deben ser XML, excepto para datos anexos binarios. Hay que notar que hoy en día uno de los formatos más usados en servicios REST en JSON.

Así, una posible definición sería hablar de ellos como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web.

Estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.

Aunque podemos encontrar diversos estilos de Servicios Web, en este estudio nos centraremos en aquellos basados en SOAP (o denominados comúnmente WS), y en REST, descartando por ejemplo otras líneas como XML-RPC [8] (considerado el precursor de SOAP).

5.3.2 SOAP WS

Un Servicio Web SOAP introduce las siguientes restricciones sobre las características ya citadas de SOA:

1. Excepto para datos binarios anexos, los mensajes deben ser transportados sobre SOAP.
2. La descripción de un servicio debe ser hecha en WSDL.
3. Uso de UDDI, que son las siglas del catálogo de negocios de Internet denominado Universal Description, Discovery and Integration.

La siguiente figura muestra cómo interactúa un conjunto de Servicios Web:

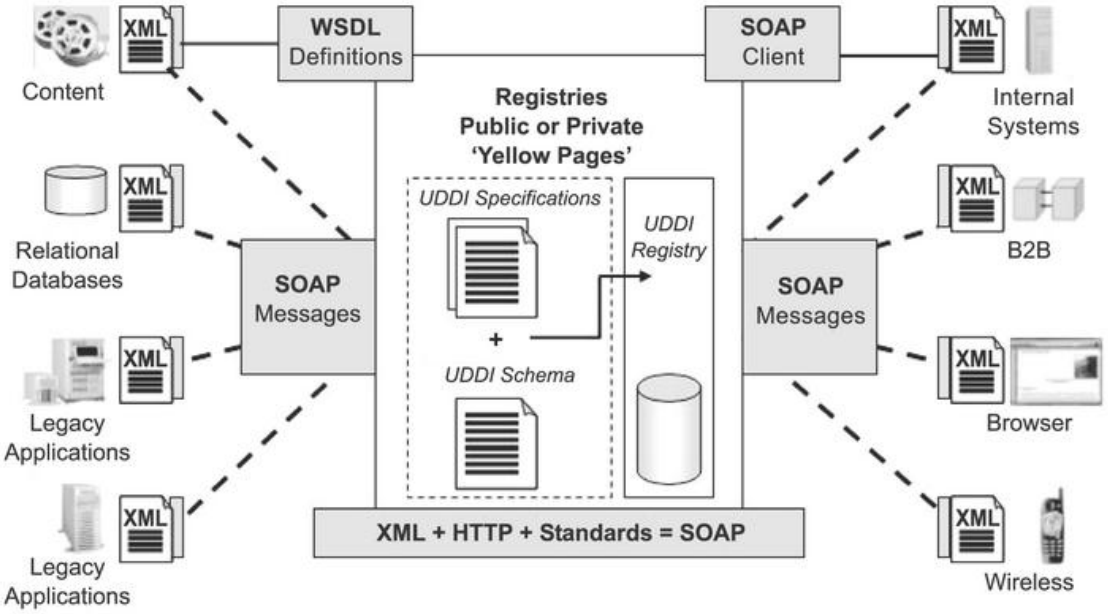


Figura 17. Esquema de interoperabilidad de servicios web basados en SOAP

Vemos que en todo este proceso intervienen una serie de tecnologías que hacen posible esta circulación de información. Un Servicio Web está formado por los siguientes componentes:

- Lógica. Se trata del componente que procesa la petición para generar la información solicitada por el cliente. Básicamente resuelve el "problema" y puede, para ello, comunicarse con otros Servicios Web, acceder a bases de datos o bien invocar API de otras aplicaciones solicitando la información (o parte de ella) que ha de generar para enviar en formato XML.

- SOAP (Simple Object Access Protocol). Protocolo de comunicación, basado en XML, que sirve para la invocación de los Servicios Web a través de un protocolo de transporte, como HTTP (ó SMTP, etc.). Consta de tres partes: una descripción del contenido del mensaje, unas reglas para la codificación de los tipos de datos en XML y una representación de las llamadas RPC para la invocación y respuestas generadas por el Servicio Web. El mensaje SOAP está compuesto por un envelope (sobre), cuya estructura está formada por los siguientes elementos: header (cabecera) y body (cuerpo).

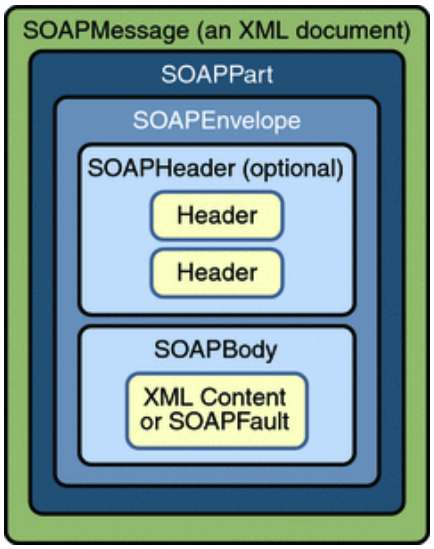


Figura 18. Estructura mensaje SOAP

- UDDI (Universal Description, Discovery and Integration): Directorio donde es posible publicar los Servicios Web, permitiendo con ello que los posibles usuarios de ese servicio puedan obtener toda la información necesaria para la invocación y ejecución del Servicio Web. Un directorio UDDI ofrece una serie de interfaces que posibilitan tanto la publicación como la obtención de información sobre los Servicios Web publicados. La información registrada se clasifica según lo que se desee obtener del servicio:
    - Información de negocio: acerca de quién publica el servicio.
    - Información de servicio: descripción del tipo de servicio.
    - Información de enlace: dirección (URL, por ejemplo) para acceder al servicio.

- WSDL (Web Services Description Language). Lenguaje basado en XML que permite la descripción de los Servicios Web definiendo la gramática que se debe usar para permitir su descripción y capacidades (datos, comandos que aceptan o producen), y su publicación en un directorio UDDI.

Durante la evolución de las necesidades de las aplicaciones basadas en Servicios Web, por ejemplo de las grandes organizaciones, se han desarrollado mecanismos que permiten enriquecer las descripciones de las operaciones que realizan sus servicios mediante anotaciones semánticas y con directivas que definen el comportamiento. Son las denominadas extensiones, que veremos de forma resumida en el siguiente apartado.

5.3.3 WS-*

La simplicidad si bien es el punto fuerte de los servicios Web, también en determinados casos puede ser su talón de Aquiles, ya que operaciones como asegurar las comunicaciones o enviar archivos binarios pueden convertirse en tareas muy complejas.

SOAP puede ser extendido realizando adiciones de módulos de funcionalidad. Este enfoque permite a los desarrolladores usar los módulos y funcionalidad que ellos necesitan, sin tener la necesidad de implementar la totalidad de estos.

Algunas de las extensiones, denominadas Ws-*, que pueden ser deseables en los proveedores son las siguientes:

- Attachments: Permite incluir documentos no XML, como archivos binarios o imágenes.
- Routing/Intermediaries: Relacionadas al proceso de enviar mensajes SOAP a través de intermediarios. Permite agregar varios Servicios Web (WS) y ofrecerlos como parte del paquete, incrementando la escalabilidad de los servicios.
- Security: Da un marco de seguridad a la comunicación. Así el SOAP envelope soporta integridad del mensaje, confidencialidad y autenticación. Describe certificados X.509, tickets Kerberos, etc.
- Quality of Services: QoS es una medida que mide la calidad del servicio, con esta extensión es posible caracterizar el funcionamiento del servicio.
- Context/Privacy: Relacionada con la WS-Security, hace referencia a guardar el contexto y privacidad, del entorno de los usuarios.

Además existen otros foros encargados de ofrecer extensiones para cubrir escenarios más complejos, o incrementar la interoperabilidad entre los distintos servicios. Los trabajos más importantes que actualmente se están llevando a cabo son:

- Interoperability: Este grupo se encarga de promover la interoperabilidad de los Servicios Web entre cada una de las plataformas, aplicaciones y lenguajes de programación. Se trata de un foro abierto donde se ofrecen a los clientes o posibles

usuarios de guías de implementación prácticas recomendadas y soporte para el desarrollo de servicios web interoperables.

- Security: trata de definir unos requisitos a aplicar en los mensajes SOAP para aumentar la seguridad en el uso de los Servicios Web garantizando la integridad de los mensajes, la confidencialidad y la autenticación.
- BPEL4WS (Business Process Execution Language for Web Services): Lenguaje para describir procesos de negocio sobre Servicios Web. Este lenguaje, combinado con las especificaciones WS-Transaction y WS-Coordination, determina como se deben de definir, coordinar e integrar los procesos de negocio dentro de la empresa o con otros proveedores, a través de sistemas heterogéneos.

En la siguiente figura se puede ver un mapa muy completo con una revisión de los estándares relacionados con los Servicios Web.

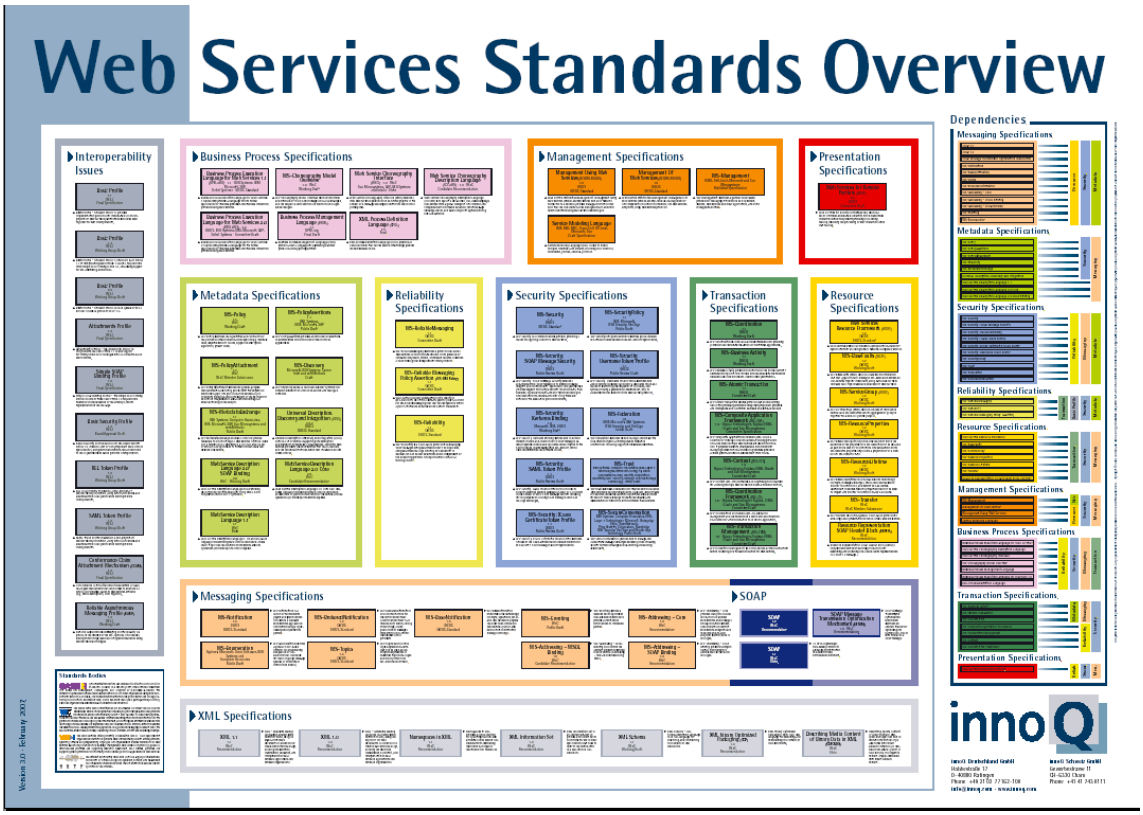


Figura 19. Estandares Servicios Web SOAP

5.4 Servicios REST

5.4.1 Introducción

El término REST, acrónimo de REpresentational State Transfer, fue introducido por primera vez por Roy Fielding (uno de los creadores de HTTP) en la lectura de su tesis para describir un tipo de arquitectura de los sistemas en red. Un Servicio Web REST es un SOA basado en el concepto de recurso. Un recurso es cualquier cosa que tiene una URI (Uniform Resource Indentifier), pudiendo tener cero o más representaciones.

Un Servicio Web REST tiene las siguientes características:

- Las interfaces deben construirse sobre HTTP. Las siguientes funciones son definidas:
    - HTTP GET: Usado para obtener una representación de un recurso. Un consumidor lo utiliza para obtener una representación desde una URI. Los servicios ofrecidos a través de este interfaz no deben contraer ninguna obligación respecto a los consumidores.
    - HTTP DELETE: Se usa para eliminar representaciones de un recurso.
    - HTTP POST: Usado para actualizar o crear las representaciones de un recurso.
    - HTTP PUT: Se usa para crear representaciones de un recurso.
- La mayoría de los mensajes son XML, definidos por un esquema XML. También pueden ser en formato JSON.
- Mensajes simples se pueden codificar en las URL.
- Los servicios y los proveedores de servicios deben ser recursos, mientras que los consumidores pueden ser un recurso.

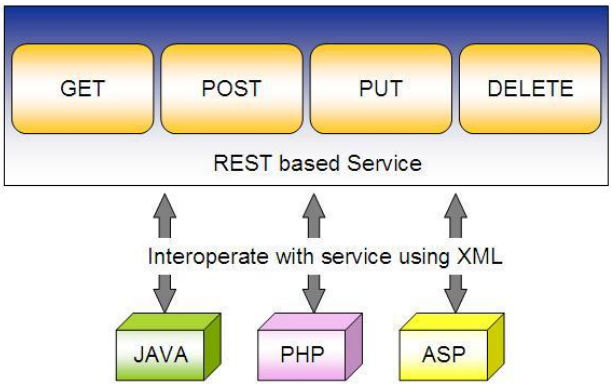


Figura 20. Esquema básico REST

Los Servicios Web REST requieren poca infraestructura, aparte de las tecnologías HTTP estándar y XML, que actualmente son soportadas por la mayoría de los lenguajes de programación y plataformas. Los servicios web REST son simples y efectivos, ya que HTTP es el interfaz más extendido y es soportado por la mayoría de las aplicaciones.

5.4.2 Arquitectura REST

La Web se compone de recursos, y un recurso es cualquier objeto de interés, identificado por una o varias URIs. Por ejemplo, una compañía de venta online puede definir un recurso "almacen", y los clientes acceder a ese recurso mediante la siguiente URL:

http://www.empresa.com/productos/almacen

La representación del recurso sería devuelta al cliente (por ejemplo, productosAlmacen.html). Esta representación sitúa al cliente en un estado. Cuando se accede a la representación del recurso, la aplicación del cliente realiza una transferencia de estado (Representational State Transfer).

La motivación para REST fue capturar las características de la Web que hacían de la propia Web unvéxito. Estas características están siendo ahora usadas para guiar la evolución de la Web. REST no es un estándar, sino que es un estilo de arquitectura (de forma análoga, no hay un estándar cliente-servidor, sino que hay un tipo de arquitectura cliente-servidor).

Sin ser un estándar, REST hace uso de estándares:

- HTTP [RFC 1945]: HyperText Transfer Protocol.
- [RFC 1738] (Uniform Resource Locator): Mecanismo de identificación de recursos.
- XML / HTML / PNG / etc.: Distintos formatos de representación de recursos.
- Tipos MIME: Como text/xml, text/html, image/png, etc.

Algunas características de REST son:

- Cliente – Servidor: Estilo de interacción basada en pull (bajo demanda).

- Sin estado: Cada petición desde un cliente hacia un servidor debe contener toda la información necesaria para comprender la petición, y no puede tomar provecho de ningún contexto almacenado en el servidor.
- Caché: Para mejorar la eficiencia de la red, las respuestas deben ser capaces de ser etiquetadas como cacheables o no cacheables.
- Interfaz uniforme: Todos los recursos son accesibles mediante un interfaz genérico (por ejemplo, HTTP GET, POST, PUT, DELETE).
- Recursos con nombres: El sistema se compone de recursos que son nombrados usando una URL.
- Representaciones de recursos interconectados: Las representaciones de los recursos están interconectadas usando URLs, por tanto un cliente está capacitado para progresar de un estado a otro.
- Componentes en capas: Intermediarios, tales como servidores proxy, servidores caché, gateways, etc., pueden ser introducidos entre los clientes y los recursos, para ofrecer rendimiento, seguridad, etc.

5.5 Ventajas y desventajas de SOAP y REST

Los Servicios Web ofrecen varios beneficios sobre otros tipos de arquitecturas de computación distribuidas:

- Interoperabilidad: Ésta es la ventaja más importante de los Servicios Web. Éstos típicamente trabajan fuera de redes privadas, ofreciendo a los desarrolladores rutas no propietarias a sus soluciones. Por tanto, los servicios suelen tener mayor vida útil, recibiendo mayor retorno ante la inversión en el servicio desarrollado. Además, los desarrolladores pueden optar por su lenguaje de programación favorito, puesto que su implementación está soportada sobre la mayoría de las tecnologías existentes. Por último, gracias al uso de métodos estándar de comunicaciones, los Servicios Web son virtualmente independientes de la plataforma.

- Usabilidad: Los servicios Web permiten que la lógica de negocio de diferentes sistemas puedan ser expuestas sobre la Web. Esto ofrece a las aplicaciones la libertad de elegir el Servicio Web que necesitan. En vez de reinventar la rueda para cada cliente, sólo es necesario incluir la lógica de negocio específica a la aplicación en el lado del cliente.

Esto permite que se desarrolle tanto el servicio como el código del lado del cliente usando los lenguajes y herramientas que se deseen.

- Reusabilidad: Los Servicios Web no ofrecen directamente un modelo de desarrollo de aplicaciones, pero casi, ya que se dispone de una aproximación que necesita un mínimo desarrollo de código para el desarrollo de dichos servicios. Esto facilita la reutilización de componentes en otros servicios.

- Despliegue: Los Servicios Web se despliegan sobre tecnologías de Internet estándar. Esto hace posible desplegar Servicios Web sobre servidores corriendo en Internet en la otra parte del globo. Además, gracias al uso de estándares, la seguridad es posible (mediante el uso de SSL por ejemplo).

Pero no todo son ventajas, podemos encontrar varios inconvenientes a tener en cuenta.

- Aunque la simplicidad de los Servicios Web es una ventaja, en algunos aspectos puede ser un estorbo. Los Servicios Web usan protocolos basados en texto plano que usan un método demasiado pesado para identificar la información. Esto significa que en ocasiones las peticiones son más largas que las peticiones codificadas con un protocolo binario. El tamaño extra es ciertamente sólo una cuestión a tratar sobre conexiones lentas, o conexiones extremadamente saturadas, pero es algo a considerar.

- Tanto HTTP como HTTPs (el núcleo de los protocolos Web) son simples, pero no fueron inicialmente considerados para sesiones largas. Típicamente, un navegador realiza una conexión HTTP, solicita una página web y después se desconecta. En entornos CORBA o RMI, un cliente se conecta al servidor, pudiendo permanecer conectado durante un periodo extenso de tiempo, recibiendo información periódica en el cliente. Esta interacción es difícil de obtener con los Servicios Web, y es necesario un trabajo extra para conseguirlo.

- El problema con HTTP y HTTPS cuando se emplean para sustentar Servicios Web es que estos protocolos son stateless, sin estado, la interacción entre el servidor y el cliente es típicamente breve, y cuando no hay datos siendo intercambiados, el servidor y el cliente no tienen conocimiento sobre el otro. Más específicamente, si el cliente hace una petición al servidor, recibirá información, y si inmediatamente cae debido a

un corte de corriente, el servidor nunca sabrá que el cliente ya no sigue activo. El servidor necesita una forma de mantener un seguimiento sobre lo que el cliente está realizando y también determinar cuándo un cliente ya no está activo.

- Típicamente, un servidor envía algún tipo de identificador de sesión al cliente cuando éste accede por primera vez al servicio. El cliente usa este identificador cuando realiza peticiones al servidor. Esto permite al servidor recuperar cualquier tipo de información que tenga sobre el cliente. Además, el servidor debería utilizar un mecanismo de timeout para determinar cuándo un cliente no está activo. Si el servidor no recibe una petición desde un cliente pasado un determinado tiempo, se asume que el cliente está inactivo y se elimina la información que se estaba manteniendo. Este overhead extra, significa mayor trabajo para los desarrolladores de los Servicios Web.

- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA, o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.

Este último hecho se acentúa en los Servicios Web sobre SOAP, ya que las cabeceras del mensaje introducen un overhead considerable, cosa que no ocurre con REST.

5.6 SOAP vs REST

Los RESTful Web Services han demostrado su capacidad para responder de manera efectiva a los requerimientos de publicación y sindicación de contenido y medios. Sin embargo, SOAP encaja mejor en soluciones con un mayor alcance y requisitos mayores, tanto en número de servicios/operaciones, número de aplicaciones cliente, número de equipos de desarrollo implicados y complejidad de mensajes, principalmente enfocados al desarrollo empresarial.

Si bien en Internet las grandes empresas están tendiendo a ofrecer sus servicios sobre REST, debido principalmente a la sencillez de invocación que ofrecen, y su limitado overhead (en los servicios sobre SOAP, las cabeceras y datos que no son la información relevante a

transmitir ocupan gran parte del ancho de banda), podemos decir que no siempre REST es la mejor opción.

Por ejemplo, REST lleva asociado HTTP como único protocolo de transporte. Si tenemos necesidad de utilizar otro tipo de transporte como JMS, SOAP sí es válido, pero no así el caso de REST. Además, en niveles de seguridad, la pila WS-* (concretamente con su definición de WS-Security a nivel de mensaje) permite un mayor nivel de seguridad que REST (que se implementa en caso de necesidad a nivel de HTTP como transporte). Las extensiones WS-* son una de las características que adolecen los servicios REST. Pero cada vez existen más iniciativas para poder cubrir estas posibles "lagunas". En relación a la seguridad, Amazon apuesta por el estándar HMAC, para poder autenticar las peticiones REST, por lo que quizás en el futuro veamos que REST puede suplir a los Servicios SOAP en todos los aspectos.

5.7 Conclusiones

Hemos visto los fundamentos y ventajas de las arquitecturas orientadas a servicios a lo largo del capítulo. También hemos comentado los diferentes tipos de servicios web asociados a este tipo de arquitecturas y tecnologías.

En el desarrollo de soluciones de sistemas de información para dispositivos móviles se hace necesaria la orientación al consumo de servicio desde una perspectiva SOA básica. Las características fundamentales asociadas a este tipo de dispositivos comentadas en capítulos anteriores justifican este enfoque, como son: limitación de capacidad de cómputo, limitación de memoria y alta conectividad.

Actualmente el tipo de servicio web más usado en este tipo de dispositivos es el servicio de tipo REST, en gran parte gracias a su simplicidad respecto de los servicios web basados en SOAP. También es posible consumir servicios web basados en SOAP, pero es una tarea muy complicada, no solo por su complejidad asociada, sino también por la ausencia de APIs de programación orientadas a este tipo de servicios en los sistemas operativos de este tipo de dispositivos. En este ámbito, la apuesta por REST no da lugar a debate y es definitiva.

A lo largo del siguiente capítulo se entrará en el detalle de la implementación de la solución propuesta, en la que existen diversos servicios web basados en REST.

6 Planteamiento de la solución

6.1 Introducción

En este capítulo se trata de dar una visión detallada de la implementación del prototipo asociado al proyecto. Es por tanto el que se desprende de la parte práctica, e intenta dar una visión global al lector, del desarrollo llevado a cabo a lo largo de la construcción del prototipo.

En el capitulo 2, planteamiento del problema, se plantearon una serie de problemas en los diversos modelos de negocio concernientes a la distribución, venta y canjeo de tickets (entradas, descuentos, etc.). Después se aportó una solución particular a la que se le puso el nombre de BeepVip. El prototipo realizado se centra en dar forma a esta solución teórica, centrándose en lo que se consideran las partes más vitales de la plataforma.

Recordemos los elementos esenciales de la arquitectura de BeepVip, presentados en la figura 1 de este documento. Por un lado se encuentra la aplicación móvil que hace uso de las funcionalidades NFC del teléfono móvil. Ésta necesita de una estructura de servicios REST alojados en servidores para consumo de datos y envío de información a la base de información central de la plataforma. En ella se encuentra gran parte de la lógica de negocio y sus reglas de negocio asociadas. Por último, está la aplicación de escritorio, que también usa servicios de los servidores y además posee de un lector NFC para establecer la comunicación necesaria con el dispositivo móvil para la realización del check-in.

Por último hay que dejar claro que la solución planteada es un prototipo, es decir, en cierto modo es incompleta. Se centra en solucionar los aspectos más delicados, con más riesgo y más novedosos que requiere la plataforma planteada, entendiendo que una vez llegado a este punto en el que el prototipo está terminado el resto de trabajo de desarrollo, que es bastante, es trivial y no supone de mucho esfuerzo creativo, documental o de análisis.

En primer lugar, en el apartado siguiente, veamos el análisis llevado a cabo.

6.2 Análisis

El análisis que se pretende llevar a cabo antes de empezar la implementación del prototipo puede parecer en un principio bastante singular. Algunos pensarán incluso que es inapropiado, sin embargo está más que fundamentado. Lo primero que hay que decir es que se pretende seguir una metodología de desarrollo ágil. Por tanto, trataremos el desarrollo de modo incremental, consiguiendo con cada iteración más consistencia en el prototipo a desarrollar. En cada iteración se marcarán determinadas metas a desarrollar, procediendo al análisis en sí solo en caso de ser absolutamente necesario para el correcto desarrollo y la consecución de la meta.

El primer problema que se presenta al inicio del desarrollo es una consecuencia lógica de la propia definición de BeepVip. La filosofía de BeepVip es la de proveer una aplicación común para la gestión y canjeo de descuentos, entradas, flyers, etc. Es por ello que se puede considerar una aplicación modular, puesto que, por ejemplo, la consulta y consumo de la información relativa a flyers de discotecas dista mucho de la que sería más adecuada para entradas de cine, y por tanto existe la necesidad de tratar la información de un modo distinto, con interfaces distintas, en lo que llamaremos módulos. El usuario debería poder elegir los módulos que le interesan y descartar los que no le interesan, ya que no le aportan nada. Sin embargo para que todos los posibles módulos coexistan en una aplicación común (BeepVip), es necesario identificar los elementos comunes existentes, es decir, extraer aquello que puede ser aplicado a todos los supuestos. Una vez extraído ese núcleo común, podremos definir la interfaz principal de la aplicación y sus funciones principales para así conseguir una aplicación coherente y consistente, además de usable.

Para conseguir identificar el núcleo principal que es inherente a todos los posibles módulos que pueden coexistir en BeepVip, se empieza por listar una cantidad importante de estos posibles módulos. De este modo podemos trabajar sobre varios supuestos y ver si realmente el núcleo principal se encuentra en todos. Los posibles tickets de los módulos o supuestos en los que se trabaja son:

1) Flyer de discoteca
2) Entrada a un cine
3) Descuento en restaurante

4) Sistema de tickets de comidas para empleados de una empresa
5) Entrada a un concierto o evento deportivo
6) Feria de la tapa
7) PoliBienvenida
8) Feria de Valencia (atracciones)
9) Descuento en una tienda de ropa
10) Oferta de viaje
11) Oferta de un centro de belleza
12) Pack de entradas a lugares turísticos de una ciudad (museos, zoo, …)
13) Descuentos al repostar combustible
14) Descuentos de un hipermercado
15) ViñaRock

Tras analizar cada supuesto nos damos cuenta de que existe una estructura base común entre todos los supuestos, que es el sistema de canje de las ofertas. Por tanto tenemos lo que buscábamos.

Sin embargo, existen diferencias en otros aspectos propios de cada tipo de oferta. Es decir, al fin y al cabo, cada oferta se compone de un conjunto de tickets. Cada ticket corresponde a un check-in. Pero cada ticket dependiendo de qué tipo de ticket sea necesita de una información específica distinta del resto. Por ejemplo, si se trata de un ticket para entrar a un cine, necesitamos información de la película, de las localidades disponibles, opiniones de la película, etc.

Queda claro, por tanto, que existe una estructura base bastante sólida y común, pero que en el elemento "ticket" existe un gran conjunto de especializaciones ya que cada tipo de ticket requiere de una información y un tratamiento distinto.

A la hora de modelar el análisis se opta por un diagrama entidad-relación en lugar de un diagrama de clases. Esto se debe a que realmente nos encontramos en una arquitectura orientada a servicios, y no a objetos. La capa de servicios recoge la lógica de negocio de la plataforma y sus reglas, pero no tiene por qué tener un enfoque propio OO, ya que se tratan de servicios REST. Por tanto resulta más fácil identificar las entidades y sus relaciones de negocio mediante este diagrama, y conseguir lo que se pretende.

Veamos un ejemplo de lo que podría ser a grosso modo la estructura de datos entidad-relación de las ofertas del tipo “locales nocturnos” y “entradas de cine”.

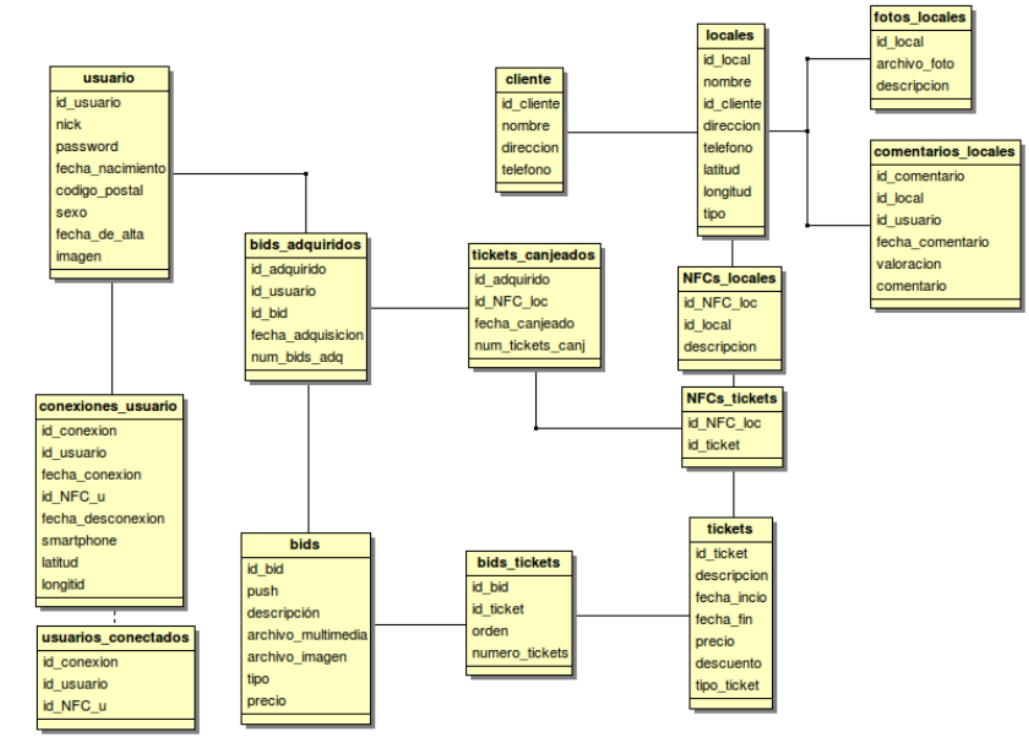


Figura 21. Entidad-relación del supuesto flyers discotecas

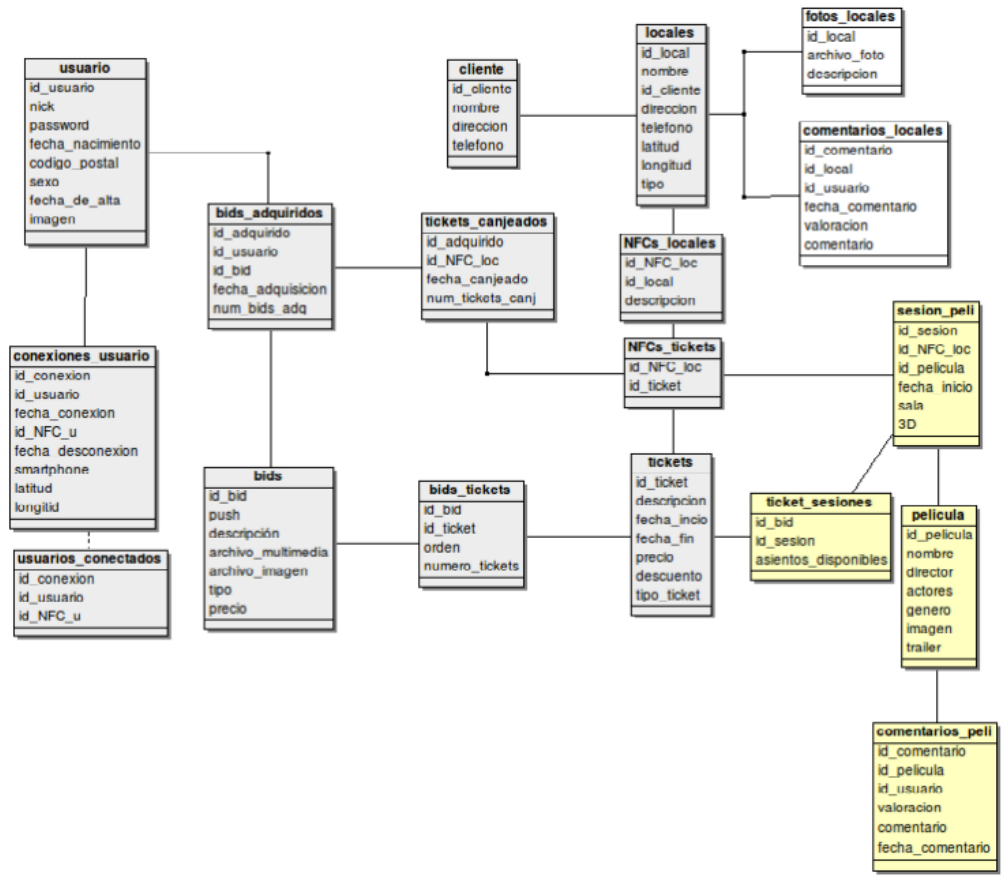


Figura 22. . Entidad-relación del supuesto entradas de cine

Si nos damos cuenta las tablas pintadas en gris son exactamente iguales a las del ejemplo anterior y corresponden con la estructura base del sistema de tickets y ofertas para los ckeck-in, como antes se había indicado. Sin embargo existen tablas en color amarillo propias del segundo ejemplo, que corresponderían con la información adicional necesaria correspondiente a la especialización del concepto ticket.

Por otro lado tenemos unas tablas en color blanco que son las que pertenecen al ejemplo de locales nocturnos, per que quizá no tendrían sentido en el ejemplo de cines.

Una vez hecho el análisis principal, podemos empezar a desarrollar la aplicación del teléfono móvil.

6.3 La aplicación del dispositivo móvil

6.3.1 Introducción

En primer lugar se elige para la implementación del prototipo el sistema operativo Android, principalmente por ser el que da más soporte a NFC. Otra consecuencia de pretender usar las funcionalidades más avanzadas de NFC en la aplicación, es la elección del API de programación. En este caso debe de ser API 14 o superior, puesto que sino no tendremos disponible lo que Android llama con el nombre de Beam, que permite comunicación P2P entre dispositivos NFC y una experiencia de usuario mucho más transparente. El API elegido para este prototipo ha sido el API 14, correspondiente a la versión Ice Cream Sandwich 4.0, siendo el SDK mínimo también el 14, lo que significa que ningún dispositivo Android con una versión del sistema operativo inferior a la elegida podrá correr la aplicación. Esto supone un gran inconveniente a la hora de intentar que la aplicación abarque la mayor masa social posible, lo que demuestra que la tecnología NFC tiene todavía un gran recorrido por delante.

6.3.2 Dispositivo móvil elegido: Samsung Google Nexus S

Como consecuencia de estar forzando a la elección de un API tan alta, y de que las funcionalidades NFC no se pueden emular en el emulador de Android para eclipse, hay que

elegir un modelo de dispositivo con capacidades NFC (no son muchos en el mercado) y con capacidad de tener instalado Ice Cream Sandwich.

El dispositivo elegido para testear el desarrollo y poder comprobar que todo funcionaba correctamente ha sido el Samsung Google Nexus S, cuya principales características son:

Características del Samsung Google Nexus S y especificaciones técnicas

- Procesador Hummingbird (ARM Cortex A8) de 1 GHz
- SO Android 2.3 Gingerbread
- 512 MB de memoria interna, 16 GB de memoria externa
- Chip NFC
- Giroscopio en 3 ejes
- Pantalla táctil capacitiva Super AMOLED de 4 pulgadas con una resolución WVGA de 800 x 480 pixels y 16.7 millones de colores, sistema anti huellas y cristal contorneado
- Cámara de fotos de 5MP con autofocus, touch focus, flash LED, geo etiquetado, tecla virtual (en pantalla) de disparo, grabación de video a 720 x 480 pixels y a 30 FPS, cámara frontal de 0.3 MP
- Reproductor de video con soporte para formatos MPEG4, WMV, H263, H264 y DivX
- Reproductor musical son soporte para portada de album y formatos MP3, FLAC, AC3, WAV y eAAC+
- Tonos de llamada polifónicos, MP3 y WAV
- Mensajería: SMS, MMS, Email, push email, mensajería instantánea, Google Talk
- Conectividad: Bluetooth 2.1 con EDR y A2DP, WiFi, USB 2.0 (micro USB), GPRS, EDGE, 3G HSDPA, HSUPA, salida de audio de 3.5 mm, sincronización con PC, GPS guia con voz integrado a Google Maps, USB y WiFi tethering
- Navegador web con soporte para contenido flash 10.1 y multi-toque
- Redes: Cuatribanda 2G GSM (850, 900, 1800, 1900) – Tribanda 3G UMTS (850, 1700, 2100)
- Batería: Ion-Litio de 1500 mAh
- Duración de la batería: 6 horas en 3G, 14 horas en GSM, 427 horas en standby 3G, 713 horas en standby GSM
- Dimensiones: 12.39 x 6.3 x 1.09 cm
- Peso: 129 g

Figura 23. Características Samsung Google Nexus S

Como podemos observas, la versión del sistema operativo por defecto es la 2.3, por lo que ha habido que instalar la versión 4 de manera no oficial para poder trabajar con él.

Figura 24. Samsung Google Nexus S

6.3.3 Configuración del entorno

El entorno elegido para el desarrollo ha sido eclipse, tal y como recomienda Google. Para poder desarrollar aplicaciones Android en eclipse hay que realizar una serie de pasos.

En primer lugar instalamos eclipse SE (Standard Edition), asegurándonos de tener instalados tanto el JRE como el JDK adecuado.

Luego, desde eclipse instalamos el pluguin ADT (Android Development Tools) mediante la dirección https://dl-ssl.google.com/android/eclipse/

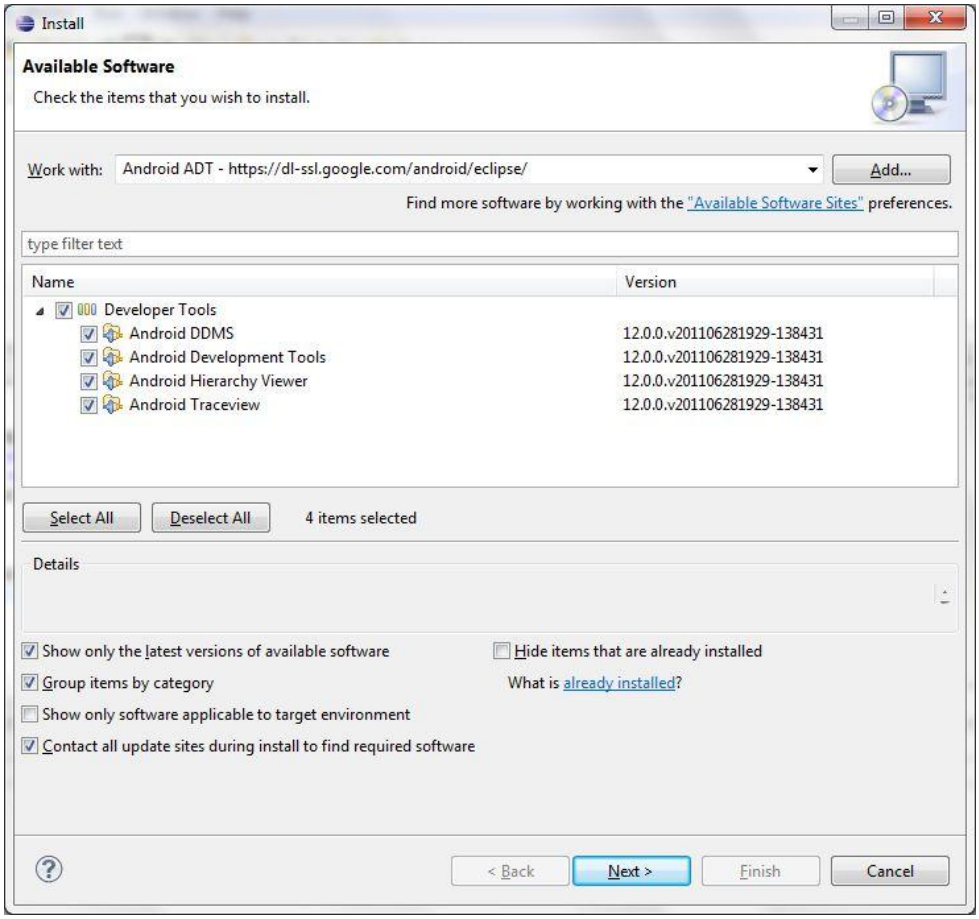


Figura 25. Instalación ADT en eclipse

Después descargamos el SDK (Software Development Kit) de Android y lo instalamos en nuestra máquina, eligiendo las APIS que de las que queremos disponer. En nuestro caso nos bastará con descargar la API 14.

Debemos configurar eclipse para que sepa donde tenemos descargado nuestro SDK Android en nuestra máquina, para ello utilizamos el SDK Manager de eclipse.

Figura 26. SDK Manager de eclipse

Luego desde eclipse vamos al botón del AVD (Android Virtual Device) Manager, en el que configuraremos un emulador Android donde ejecutar nuestros desarrollos sin la necesidad de pasarlos a un dispositivo físico. Como ya se ha dicho antes, en determinadas ocasiones el emulador de Android no es la opción correcta para probar nuestras desarrollos, como es el caso del uso de NFC.

Figura 27. Emulador Android

Ahora ya está todo listo para empezar el desarrollo.

6.3.4 Activities (interfaces)

El prototipo desarrollado consta de un total de 23 clases java, de las cuales 9 extienden de la clase Activity. Como ya se comentó en el capítulo de Android, una Activity es el componente

básico de una aplicación ya que refleja una determinada actividad llevada a cabo por una ésta, y lleva asociada típicamente una ventana o interfaz de usuario.

De las 9 activities que tiene el prototipo, 7 son interfaces de usuario, 2 son ventanas con mensajes de alerta y una es una clase abstracta para la implementación de interfaces con listas. De las 7 interfaces, 6 pertenecen al núcleo principal del BeepVip, es decir a los elementos comunes entre los módulos que se desprendieron del análisis del apartado anterior de este capítulo, y una a un módulo en particular, que en este caso es FeverNight (flyers de locales de ocio nocturno).

La primera interfaz con la que nos encontramos al ejecutar por primera vez la aplicación es la que se desprende de la clase java BeepVipActivity. Esta interfaz se muestra a continuación:

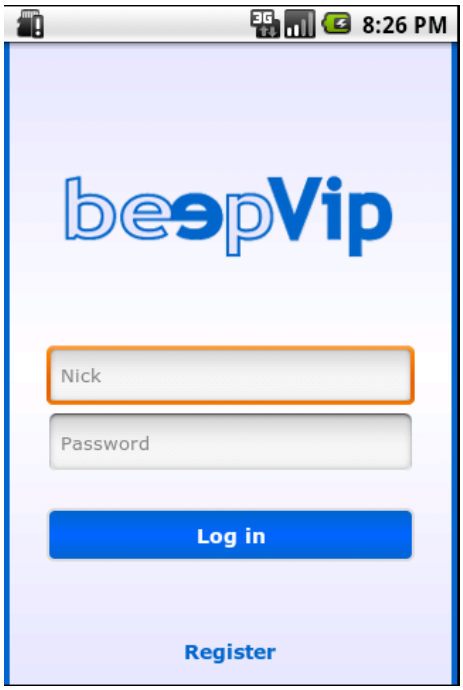


Figura 28. Interfaz de la activity BeepVipActivity

En ella podemos ver el logo principal y dos campos a rellenar por el usuario (Nick y Password), además de un botón "Log in" para poder loggearse en la aplicación. También se puede ver un enlace en la parte inferior para registrarse en el caso de no ser usuario de BeepVip, que nos llevará a otra interfaz.

Si el usuario introduce sus credenciales con éxito, se mostrará la interfaz principal de la aplicación, que más adelante veremos. Si sus credenciales no son válidas se mostrará un

mensaje de alerta para avisar al usuario. Cuando un usuario se loggea correctamente, se guarda su "id" en un archivo de preferencias Android (relaciones clave-valor) en la clave "usuario". De este modo si el usuario sale de la aplicación sin desloguearse cuando vuelva a entrar entrará directamente en la pantalla principal.

Para saber si el usuario pertenece o no al conjunto de usuarios BeepVip se hace uso de un servicio REST que lleva por nombre "log", que devuelve un el número de id del usuario, o 0 en caso de no haber introducido bien las credenciales. El consumo del servicio y su tratamiento mediante código se muestra en la siguiente figura.

```
public boolean validarLog(){

    HttpClient httpClient = new DefaultHttpClient();
    HttpPost post =
        new HttpPost("http://192.168.43.186:8080/BeepVip_WS/rest/ws/log");
    post.setHeader("content-type", "application/json");
    JSONObject dato = new JSONObject();
    TelephonyManager mTelephonyManager;
    mTelephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
    String imei=""+mTelephonyManager.getDeviceId();

        try {
        dato.put("nick", et1.getText().toString());
        dato.put("pass", et2.getText().toString());
        dato.put("imei", imei);
        StringEntity entity = new StringEntity(dato.toString());
        post.setEntity(entity);
        HttpResponse resp = httpClient.execute(post);
        String respStr = EntityUtils.toString(resp.getEntity());
        if(!respStr.equals("0")){
            SharedPreferences id_usuario=getSharedPreferences(usuario, 0);
            SharedPreferences.Editor editor=id_usuario.edit();
            editor.putInt("id", Integer.parseInt(respStr));
            editor.commit();
            return true;
        }
```

Figura 29. Consumo y tratamiento del servicio "log"

Si el usuario no está registrado, pasará a la interfaz de registro de usuario (RegistrarActivity). Esta actividad no es más que un formulario en el que se le solicitan al usuario determinados datos de interés para la plataforma. Estos datos personales junto con el nick y el password serán usados por las reglas de negocio para ofrecer tickets selectivos a los usuarios. A continuación se muestra la interfaz de registro de usuario.

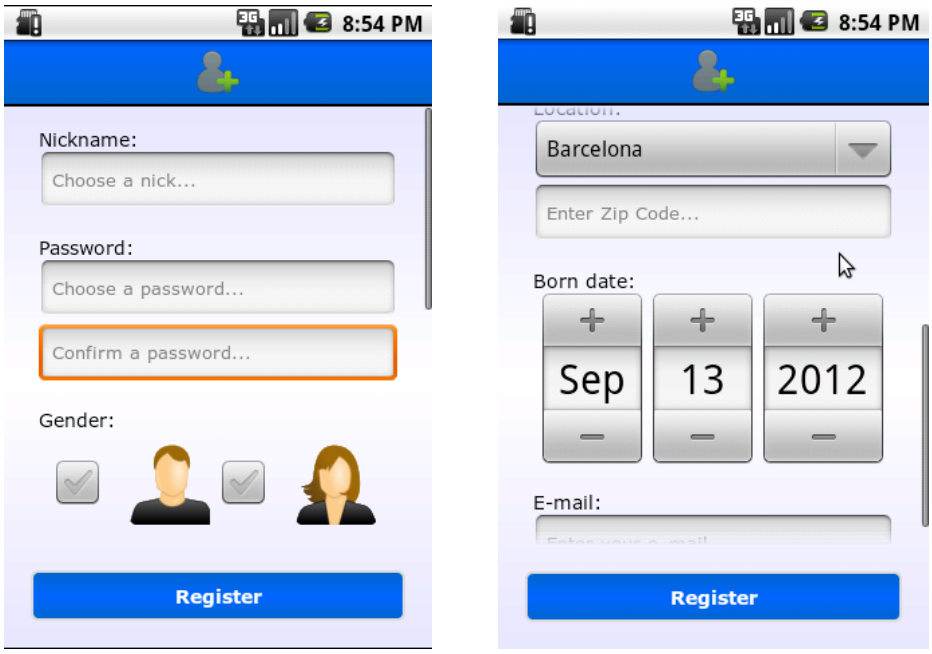


Figura 30. Interfaz de registro de usuario

Cuando el usuario pincha en el botón “register”, se ejecuta un método que valida que los datos están introducidos correctamente. Si no están introducidos correctamente se mostrará un mensaje de alerta que informará al usuario de los campos mal rellenados. Si están bien introducidos llamará al servicio REST “add” que se ocupará de almacenar al usuario en la base de datos de la plataforma. Seguidamente se mostrará la interfaz principal.

Figura 31. Interfaz principal BeepVip

En la interfaz de la figura anterior podemos ver algunos ejemplos de los posibles módulos que puede albergar BeepVip, en este caso; FeverNight (flyers de locales de ocio nocturno), Cinema (entradas de cine) y Events (entradas a eventos). En la parte inferior vemos 4 botones que son la base del sistema de tickets BeepVip. El primero de ellos es el botón social (no está implementado) en el cual un usuario puede compartir tickets con su red social de amigos BeepVip. El segundo es el botón de gestión de tickets, mediante el cual el usuario puede entrar a una interfaz en la que gestiona los tickets que tiene descargados; los puede eliminar, compartir, ver información del local, hora, descuento, etc. El tercer botón es el botón de tickets VIP. Este es un botón que lleva al usuario a visualizar los tickets VIP que le han sido mandados. Un ticket VIP puede ser un ticket de cualquier clase (módulo) que le ha sido enviado selectivamente a un pequeño grupo de usuarios atendiendo a varios criterios establecidos por la empresa que ofrece el ticket. Por último está el botón de check-in, que es el que el usuario utilizará para canjear un ticket en un determinado local a través de NFC. Además en la parte superior podemos ver el botón de configuración, en el que configurar diversos aspectos de BeepVip, como por ejemplo los módulos a los que se desea tener acceso.

Hay que notar que los botones inferiores están en blanco y negro, lo que indica que no están habilitados. Esto sucede porque el usuario no tiene ningún ticket pendiente de canjeo y tampoco tiene ningún ticket VIP pendiente. Cuando los un botón de la barra inferior está activo su icono está en color.

Imaginemos que un usuario quiere ver los descuentos que le ofrece uno de los módulos, en este caso el módulo FeverNight que es el que está implementado. La interfaz de FeverNight muestra una lista de flyers válidos para el usuario, que pueden ser mostrados mediante diversos filtros.

La lista se obtiene del consumo de un servicio REST que devuelve un conjunto de tickets que son almacenados en una base de datos SQLite 3 de Android. Los tickets se almacenan al consumir el recurso para poder tratarlos de una mejor forma a la hora de navegar entre las distintas opciones de filtrado, pero una vez el usuario vuelve a la interfaz principal la base de datos es borrada para no consumir almacenamiento innecesariamente.

Los filtros existen para ese módulo se muestran en una barra horizontal desplazable (Horizontal ScrollBar), y en este caso se pueden mostrar los tickets: ordenados por destacados, solo los de discotecas, solo los de pubs, por cercanía, etc.

Cuando al usuario le interesa un ticket puede pinchar sobre su ítem de la lista, y se desplegará un menú en la parte inferior del ítem que permitirá ubicar en un mapa el local, compartir el ticket o descargarse el ticket. Sólo esta implementada la función de descarga de ticket.

Al descargar el ticket se vuelve a la interfaz principal y se puede ver como el botón de tickets del menú inferior ha cambiado a color y tiene un número al lado. Ese número indica el número de tickets descargados por el usuario. El ticket descargado se guarda en una base de datos SQLite para que aunque el usuario se salga de la aplicación y vuelva a entrar sin tener conexión a internet, la aplicación sepa los tickets disponibles que tiene y pueda canjearlos.

Si el usuario pincha en el botón de gestión de tickets, verá una interfaz con un listado de los tickets descargados, si picha sobre un ítem de la lista se desplegará un menú muy similar al que se desplegaba en FeverNight, con la única diferencia que en lugar de estar el botón de descargar ticket, está el botón de eliminar ticket, mediante el cual se eliminaría el ticket de la base de datos de tickets descargados y no se podría canjear.

Otra forma de descargar un ticket, en lugar de a través de los módulos, es a través de lo tickets VIP. Existe un “service” de tickets vip en la aplicación. Un service es un procedimiento en segundo plano que se puede iniciar y parar. El service de BeepVip se inicia en el momento que un usuario se loggea y pregunta la servidor cada cierto tiempo (30 segundos en el prototipo) si hay nuevos tickets VIP para un determinado usuario. Si la respuesta es afirmativa se lanza una notificación Android al usuario para indicarle que tiene nuevas ofertas VIP. Si el usuario pincha sobre la notificación se abrirá la interfaz principal de BeepVip en la que el número asociado al botón de VIPS se habrá incrementado en tantos tickets VIP nuevos disponga el usuario. Si el usuario pincha sobre este botón podrá ver un listado de las ofertas VIP, y de forma análoga a lo visto en el módulo FeverNight podrá descargar una oferta.

Por último, solo queda por analizar la interfaz de canjeo de tickets. Cuando un usuario tiene tickets descargados y quiere canjearlos pincha sobre el botón de check-in y acerca su terminal al lector NFC del local. En ese momento se produce la comunicación NFC entre el dispositivo y el reader del local, que puede derivar en un ticket canjeado.

6.3.5 Implementación comportamiento NFC

El comportamiento NFC del check-in en la aplicación se implementa usando el api correspondiente sobre la manipulación y control de esta tecnología. A continuación se muestra el código necesario para ello, que posteriormente será comentado.

```
1  package org.upv.es;
2
3  import java.nio.charset.Charset;
18
19 public class NfcActivity extends Activity{
20
21     private NfcAdapter mAdapter;
22     private boolean procesado=false;
23
24     /** Called when the activity is first created. */
25     @Override
26     public void onCreate(Bundle savedInstanceState) {
27         super.onCreate(savedInstanceState);
28         setContentView(R.layout.nfc);
29         mAdapter = NfcAdapter.getDefaultAdapter(this);
30
31     }
32
33     @Override
34     protected void onPause() {
35         // TODO Auto-generated method stub
36         super.onPause();
37         if(procesado){
38             finish();
39         }
40         mAdapter.disableForegroundNdefPush(this);
41     }
42
```

Figura 32. Código NFC Android

La implementación del comportamiento NFC deseado está estrechamente ligada al ciclo de vida de la "activity". Cuando se crea la instancia se llama al método onCreate(), que no volverá a ser llamado. Es entonces cuando creamos el adaptador NFC, que lo dejamos como inactivo en onPause(), ya que la actividad no está en vistas de estar activa. Es también en este método donde comprobamos si el check-in ya ha sido procesado con la intención de saber si tenemos que seguir en la interfaz.

Luego, como se observa en la siguiente figura se llama a onResume(). En este método configuramos el NFC de tal manera que le decimos el mensaje que debe de trasmitir, y se le deja activo a la espera de recibir respuesta. Cuando reciba respuesta llamará a onNewIntent, en el que se prepara la información para analizar la respuesta con el método checkstatus().

```
@Override
protected void onResume() {
    // TODO Auto-generated method stub
    super.onResume();
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
            new Intent(this, this.getClass())
                    .addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP), 0);
    mAdapter.enableForegroundDispatch(this, pendingIntent, null, null);
    SharedPreferences usuario=getSharedPreferences("MisPreferencias", 0);
    NdefRecord recordToSend = createRecord(String.valueOf(usuario.getInt("id", 0)));
    NdefMessage messageToSend = createMessage(recordToSend);
    mAdapter.enableForegroundNdefPush(this, messageToSend);
}

@Override
public void onDestroy() {
    super.onDestroy();
}

@Override
protected void onNewIntent(Intent intentNFC) {
    // TODO Auto-generated method stub
    super.onNewIntent(intentNFC);
    String action = intentNFC.getAction();
    checkstatus(action,intentNFC);
}
```

Figura 33. Código NFC Android (II)

Cuando se ejecuta checkstatus se analiza el mensaje recibido. Si el mensaje es distinto de 0 es porque dicho mensaje contiene el id del ticket a canjear. Se procede al canjeo del ticket, que consiste en borrar de la base de datos el ticket en cuestión e informar al usuario de que ha sido canjeado. Si el usuario recibe como respuesta un 0, se le indica que no tiene el ticket necesario para entrar al local.

```
public void checkstatus(String action, Intent intentNFC) {
    if (NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
        Parcelable[] rawMsgs = intentNFC
                .getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
        ArrayList<NdefMessage> messages = new ArrayList<NdefMessage>();
        if (rawMsgs != null) {
            for (int i = 0; i < rawMsgs.length; i++) {
                messages.add((NdefMessage) rawMsgs[i]);
                NdefRecord record = messages.get(i).getRecords()[i];
                String message = getTextData(record.getPayload());
                if(!message.equals("0")){
                TicketsSQLite db=new TicketsSQLite(this);
                db.canjearTicket(message);
                db.close();
                this.mostarMensajeCanjeado();
                }else{
                    this.mostarMensajeCancelado();
                }
            }

        }
    }
}
```

Figura 34. Código NFC Andrroid (III)

6.4 La aplicación de escritorio

6.4.1 Introducción

El objetivo de la aplicación de escritorio de la plataforma BeepVip es la de dotar a los locales asociados de la capacidad de gestionar el canjeo de tickets de una forma automática. Para ello se cuenta con una aplicación desarrollada en Java, una base de datos MySQL y un lector NFC.

El grueso del trabajo de desarrollo de esta parte de la arquitectura BeepVip reside en la puesta en marcha del reader NFC para que pueda ser usado por la aplicación, tarea que no es en absoluto trivial.

El trabajador/a de la empresa utilizará la aplicación para saber si un determinado cliente posee un ticket válido para ese momento, y podrá saber bajo qué condiciones el cliente adquirió el ticket, pudiendo cobrar arreglo a esas condiciones.

Por ejemplo, si una discoteca ha ofrecido descuentos selectivos a diferentes perfiles de usuario BeepVip y un usuario ha descargado ese ticket y se dispone a canjearlo, el usuario acercará su dispositivo al reader NFC del local, y la aplicación de escritorio mostrará el precio que debe pagar por entrar y las condiciones de la entrada. La aplicación podría mostrar que el usuario debe pagar 4€ por entrar y tiene una consumición gratis, por ejemplo.

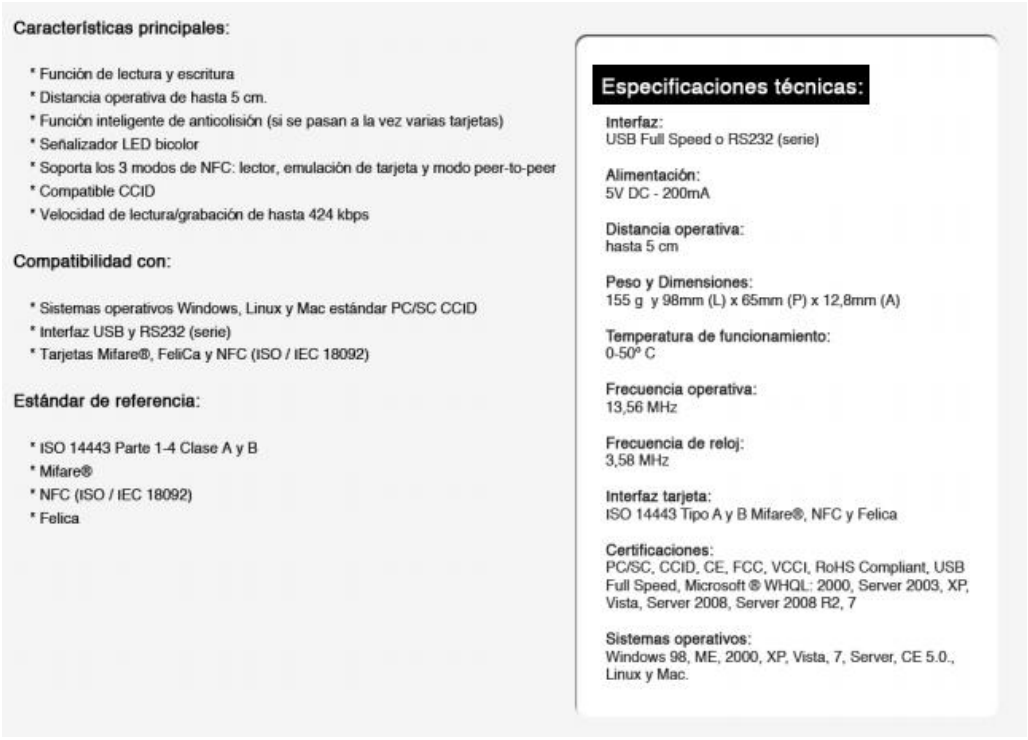
6.4.2 El reader NFC: ACS ACR122-U

Antes que nada hay que elegir el reader NFC que queremos comprar para poder implementar el prototipo. Tras un a análisis exhaustivo de la oferta existente en este mercado se decide comprar el reader NFC de la marca ACS, en concreto el modelo ACR122-U.

Los motivos de la elección de este reader en concreto son varios. Por un lado su precio es bastante asequible, unos 70€ con gastos de transporte incluidos. Además debido a sus características es un reader de los más usados entre la comunidad de desarrolladores NFC, lo que implica que se dispone de más información y feedback que del resto de readers del

mercado. Como consecuencia de esto último, también se da más soporte en las diferentes apis open source existentes, algo que es capital a la hora de llevar a cabo el desarrollo, puesto que se ha optado por comprar el reader sin el SDK asociado que ofrece la empresa. Si se hubiera optado por la compra de dicho SDK el trabajo de desarrollo se hubiera simplificado notablemente.

Las características principales del ACS ACR122-U son las que se muestran en la siguiente figura.



Características principales:

* Función de lectura y escritura
* Distancia operativa de hasta 5 cm.
* Función inteligente de anticolisión (si se pasan a la vez varias tarjetas)
* Señalizador LED bicolor
* Soporta los 3 modos de NFC: lector, emulación de tarjeta y modo peer-to-peer
* Compatible CCID
* Velocidad de lectura/grabación de hasta 424 kbps

Compatibilidad con:

* Sistemas operativos Windows, Linux y Mac estándar PC/SC CCID
* Interfaz USB y RS232 (serie)
* Tarjetas Mifare®, FeliCa y NFC (ISO / IEC 18092)

Estándar de referencia:

* ISO 14443 Parte 1-4 Clase A y B
* Mifare®
* NFC (ISO / IEC 18092)
* Felica

Especificaciones técnicas:

Interfaz:
USB Full Speed o RS232 (serie)

Alimentación:
5V DC - 200mA

Distancia operativa:
hasta 5 cm

Peso y Dimensiones:
155 g y 98mm (L) x 65mm (P) x 12,8mm (A)

Temperatura de funcionamiento:
0-50º C

Frecuencia operativa:
13,56 MHz

Frecuencia de reloj:
3,58 MHz

Interfaz tarjeta:
ISO 14443 Tipo A y B Mifare®, NFC y Felica

Certificaciones:
PC/SC, CCID, CE, FCC, VCCI, RoHS Compliant, USB Full Speed, Microsoft ® WHQL: 2000, Server 2003, XP, Vista, Server 2008, Server 2008 R2, 7

Sistemas operativos:
Windows 98, ME, 2000, XP, Vista, 7, Server, CE 5.0., Linux y Mac.

Figura 35.Características ACS ACR122-U

Lo que más nos interesa de todas sus características es el ofrecimiento de soporte del modo NFC P2P (peer-to-peer), que es el que usaremos en el prototipo.

Figura 36. ACS ACR122-U

6.4.3 Funcionamiento

El funcionamiento de esta aplicación de escritorio, a excepción de lo que concierne a NFC, es bastante simple. La implementación de la solución consta de tan solo de dos clases Java, y una de ellas es exclusiva del tratamiento NFC.

Para el desarrollo de la interfaz principal (única interfaz de la aplicación), se ha usado el pugling Jigglo de desarrollo de interfaces en Java en su versión no comercial. La interfaz simplemente muestra cuando un usuario ha canjeado un ticket el nick del usuario, las condiciones implícitas al ticket y el precio que se le debe de cobrar. En caso de no ser un canjeo valido, se muestra en la interfaz un mensaje avisando al empleado del local que dicho usuario no está en lista.

En la capa de persistencia de la aplicación nos encontramos con una base de datos MySQL de tabla única. La tabla en cuestión lleva por nombre lista, y se dedica a almacenar la lista de tickets que los usuarios han descargado de ese local. Su definición es como se muestra en la figura.

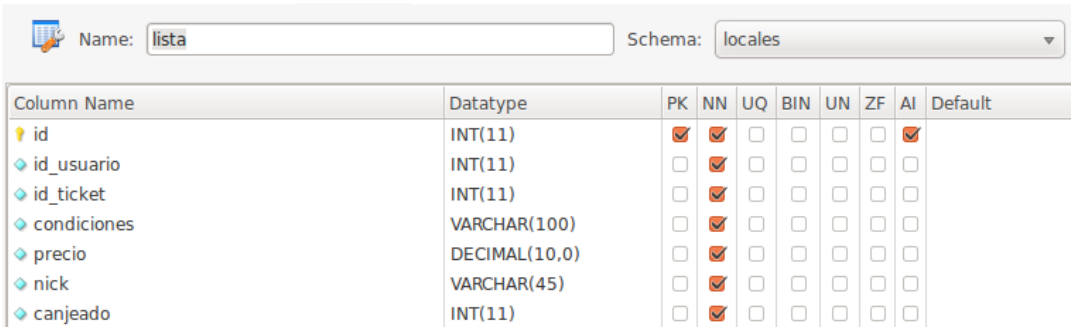


Figura 37. Tabla "lista" de la base de datos de la aplicación

Para poblar esta base de datos y tenerla actualizada se hacen llamadas a un servico REST de la capa de servicios de la plataforma que devuelve lo tickets que todavía no han sido descargados por el local. Este servicio se llama getlista, y se le ha de pasar el id del local como parámetro.

Cuando un dispositivo móvil se acerca al reader NFC de la aplicación de escritorio del local con la intención de canjear un ticket, se establece la conexión entre los dispositivos. Luego, el dispositivo móvil pasa un mensaje NDEF al reader con el id de usuario del propietario del

smartphone. El reader recibe el mensaje y llama a un método para saber si el usuario está en lista. Si el usuario está en lista lanza un mensaje con el id del ticket que tiene el usuario y lo marca como canjeado en la base de datos. Ahí acaba la comunicación NFC del sistema, y se reinicia a la espera de un nuevo usuario. Si el usuario no estuviera en lista, el mensaje devuelto sería un 0.

Veamos pues cómo se implementa la funcionalidad NFC que acabamos de decribir.

6.4.4 Implementación del comportamiento NFC

Después del todo el proceso de investigación y documentación comentado anteriormente para la elección del API NFC en Java más adecuado para el desarrollo de todas las funcionalidades pretendidas, se opta por la elección de un API que se llama nfctools, y que es desarrollada por una comunidad open source.

Las clases utilizadas del nfctools para la implantación de la solución son las que se muestran a continuación.

```
package com.beepvip.local;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.nfctools.NfcAdapter;
import org.nfctools.examples.TerminalUtils;
import org.nfctools.examples.ndef.NdefExampleRecords;
import org.nfctools.llcp.LlcpConnectionManager;
import org.nfctools.llcp.LlcpConstants;
import org.nfctools.llcp.LlcpOverNfcip;
import org.nfctools.ndef.NdefListener;
import org.nfctools.ndef.Record;
import org.nfctools.ndef.mime.TextMimeRecord;
import org.nfctools.ndef.wkt.records.TextRecord;
import org.nfctools.ndefpush.NdefPushFinishListener;
import org.nfctools.ndefpush.NdefPushLlcpService;
import org.nfctools.scio.TerminalMode;
import org.nfctools.utils.LoggingNdefListener;
```

Figura 38. Clases usadas de nfctools

En primer lugar se instancia la clase LlcpView. Al llamar al constructor, se le pasa un modo de inicio y una vista o clase de interfaz. Después se crea un servico push de formato NDEF con un listener asociado para ese tipo de mensajes. Por último se crea el conectionManager.

```
public class LlcpView extends Thread {

    private NdefPushLlcpService ndefPushLlcpService;
    private boolean initiatorMode;
    private LlcpOverNfcip llcpOverNfcip;
    private NewJFrame nfcView;

    public LlcpView(boolean initiatorMode, NewJFrame view) {
        nfcView = view;
        this.initiatorMode = initiatorMode;
        System.out.println("obtendo el modo de inicio: true->initiator  false->target  es:"+String.valueOf(initiatorMode));
        ndefPushLlcpService = new NdefPushLlcpService(new NdefListener() {

            @Override
            public void onNdefMessages(Collection<Record> clctn) {
                ArrayList<Record> listRecord = (ArrayList<Record>) clctn;
                TextRecord r = (TextRecord) listRecord.get(0);
                System.out.println("he entrado en el listener y obtengo: "+r.getText());
                nfcView.displayResultOfAnswer(r.getText());

            }
        });
        System.out.println("creo el service con su listener para onNdefMessage");
        LoggingNdefListener N = new LoggingNdefListener();
        System.out.println("creo el log");
        llcpOverNfcip = new LlcpOverNfcip();
        LlcpConnectionManager connectionManager = llcpOverNfcip.getConnectionManager();
        connectionManager.registerWellKnownServiceAccessPoint(LlcpConstants.COM_ANDROID_NPP, ndefPushLlcpService);
        System.out.println("creo el manager de conexion y registro los mensajes tipo ndef de Android");
    }
```

Figura 39. Implementación de la configuración NFC

Después se llama al método run() de la clase, que extiende de Thread. Aquí se crea el adaptador y se crea la conexión del tipo llcp. Por último se empieza la escucha, y por tanto cuando llegue un mensaje se llamará al método onNdefMessage(), que a su vez llamará a displayResultOfAnswer().

```
@Override
public void run() {
    while (true) {
        try {
            TerminalMode terminalMode = initiatorMode ? TerminalMode.INITIATOR : TerminalMode.TARGET;
            NfcAdapter nfcAdapter = new NfcAdapter(TerminalUtils.getAvailableTerminal(), terminalMode);

            System.out.println("creo el adaptador al terminal con el metodo de inicio");

            nfcAdapter.setNfcipConnectionListener(llcpOverNfcip);

            System.out.println("le metdo el listener");

            nfcAdapter.startListening();

            System.out.println("empiezo a escuchar");

            System.out.println("Mode: " + terminalMode);
            System.out.println("Waiting for P2P, press ENTER to exit");
            System.in.read();
        } catch (IOException ex) {
            Logger.getLogger(LlcpView.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Figura 40. Inicio del NFC

El método llamado analizará el mensaje que se ha recibido para saber si el usuario está en lista, y montará el mensaje oportuno para ser enviado vía NFC, o bien con el ide del ticket que dispone el usuario o bien con un 0.

```
public void displayResultOfAnswer(String text) {
    try
    {
    Class.forName("com.mysql.jdbc.Driver");
    Connection conexion = DriverManager.getConnection ("jdbc:mysql://localhost:3306/locales","root", "root");
    Statement s = conexion.createStatement();
    System.out.println("este dato paso a la selcect: "+text);
    ResultSet rs=s.executeQuery("select * from locales.lista where id_usuario= "+text+"");
    int i=0;
    while(rs.next()){
        i++;
    }
    if(i!=0){
        rs.last();
        nfcManager.addMessages(String.valueOf(rs.getInt("id_ticket")));
        jLabelNick2.setText(rs.getString("nick"));
        jLabel1.setText(String.valueOf(rs.getDouble("precio"))+" €");
        jLabelCondiciones2.setText(rs.getString("condiciones"));
        usuarioAct=text;
        valido=true;
    }else{
        nfcManager.addMessages("0");
        jLabelNick2.setText("el usuario no esta en lista");
        jLabel1.setText("error");
        jLabelCondiciones2.setText("");
        valido=false;
    }
    rs.close();
    conexion.close();
    }catch(Exception e){
        e.printStackTrace();
    }

}
```

Figura 41. Montando el mensaje de respuesta NFC

Por último se transmite el mensaje y se queda a la espera de asegurarse que el dispositivo móvil ha recibido la respuesta a través de dos listeners. Cuando esto ocurre se reinicia el Thread, quedando a la espera de un nuevo usuario.

```
public void addMessages(String message) {

    ndefPushLlcpService.addMessages(NdefExampleRecords.convertToList(new TextMimeRecord("text/ndefTransaction", message))

        @Override
        public void onNdefPushFinish() {
            System.out.println("el ndef push finish ha sido llamado");
            nfcView.sendingStatusToView(true);
        }

        @Override
        public void onNdefPushFailed() {
            System.out.println("el ndef push failed ha sido llamado");
            nfcView.sendingStatusToView(false);
        }
    });
}
```

Figura 42. Manejo del envío de respuesta NFC

Con esto ya tenemos la funcionalidad NFC implementada y lista para usarse.

6.5 La capa de servicios

Por ultimo nos queda por ver la última parte de la arquitectura BeepVip expuesta en la fugura 1. Esta parte es la capa de servicios REST junto con el backend de la plataforma, que es una base de datos MySQL.

Los servicios REST implementados son 6:

- add: se usa para registrar un nuevo usuario en la base de datos.
- log: sirve para loggear al usuario.
- getticketsfn: devuelve todos los tickets válidos que puede disponer un usuario en el módulo FeverNight.
- descargar: da a conocer al sistema que un usuario ha descargado un determinado ticket desde su dispositivo móvil.
- getlista: devuelve al local los ítems de la lista de descargas que todavía no han sido enviados.
- getvips: devuelve los tickets vip que dispone un usuario en concreto, y todavía no han sido enviados.

Para la implementación de los servicios REST en Java usamos JAX-RS. JAX-RS es una especificación de Sun (Oracle) para implementar servicios REST usando clases Java junto con anotaciones, lo que facilita bastante la tarea. La implementación de JAX-RS usada es Jersey, que debe descargarse e importas sus archivos .jar al directorio WebContent/WEB-INF/lib del proyecto. Por otro lado tenemos instalado y configurado Tomcat para servir los servicios a través del puerto 8080. Dentro de la carpeta WEB-INF/lib configuramos el servlet Jersey web.xml, que será usado por Tomcat. La configuración del servlet se hace del siguiente modo:

```
< servlet >
    < servlet - name > Jersey REST Service </ servlet - name >
    < servlet - class > com.sun .jersey.spi . container . servlet .ServletContainer
    </ servlet - class >
    < init - param >
            < param - name > com.sun.jersey.config.property.packages </ param - name >
            < param - value >es.upv.dsic.tesis </ param -value >
    </ init - param >
```

< load - on - startup >1 </ load - on - startup >

</ servlet >

< servlet - mapping >

< servlet - name > Jersey REST Service </ servlet - name >

<url - pattern >/ rest / </ url - pattern >*

</ servlet - mapping >

De este modo la implementación de un servicio REST quedaría de la siguiente forma, haciendo uso de anotaciones Java.

```
@Path("/ws")
public class BeepVip_WS {

    @Path("/add")
    @POST
    @Consumes("application/json")
    @Produces ("text/plain")
    public String registrar(Usuario usuario){
        .
        .
        .
```

Figura 43. Definición del servicio "add" con Jersey

En cuanto la persistencia usada por la capa de servicios, es decir, el backend de la plataforma, se ha usado MySQL como motor de la base de datos. En total hay una base de datos con 4 tablas: usuarios, sesiones, tickets_fn y descargas.

Name: tickets_fn Schema: beepvip_bd

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id_ticket	INT(11)	☑	☑	☐	☐	☐	☐	☑	
titulo	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
descripcion	VARCHAR(150)	☐	☑	☐	☐	☐	☐	☐	
subtitulo	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
desde	DATETIME	☐	☑	☐	☐	☐	☐	☐	
hasta	DATETIME	☐	☑	☐	☐	☐	☐	☐	
hombres	INT(10)	☐	☑	☐	☐	☑	☐	☐	'0'
mujeres	INT(10)	☐	☑	☐	☐	☑	☐	☐	'0'
precio	DECIMAL(10,0)	☐	☑	☐	☐	☑	☐	☐	
id_local	INT(11)	☐	☐	☐	☐	☐	☐	☐	NULL
imagen	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
prioridad	INT(11)	☐	☐	☐	☐	☐	☐	☐	NULL
tipo	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
vip	INT(11)	☐	☑	☐	☐	☐	☐	☐	

Figura 44. Tabla de tickets disponibles

Las tablas comentadas tienen las siguientes misiones principales:

- Usuarios: guarda toda la información relativa a los usuarios del sistema
- Sesiones: almacena las sesiones iniciadas por todos los usuarios
- Tickets_fn: guarda los tickets existentes del módulo FeverNight, al no haber implementado ningún otro módulo, esta tabla guarda todos los tickets posibles.
- Descargas: almacena la información de los tickets descargados por los usuarios.

Hasta aquí ya hemos dado una visión detallada de cómo se construye el prototipo de este proyecto. Aun así se anexa todo el código.

6.6 Propuestas de mejora

No hay que olvidar que lo que se ha desarrollado en la parte práctica del proyecto es un prototipo, que pretende demostrar que el planteamiento inicial sobre un modelo de negocio propio en el ámbito del canjeo de tickets, gestión de descuentos, etc. usando tecnologías de vanguardia es totalmente posible. Al tratarse de un prototipo de estas características, las mejoras admitidas son quasi infinitas.

Por un lado habría que mejorar la profundidad de desarrollo, es decir, implementar todas las funcionalidades que se describen en la solución teórica planteada. Por ejemplo, solo se ha implementado un módulo de entre todos los posibles en BeepVip. Tampoco se ha implementado la función social de compartir tickets entre amigos BeepVip.

Por otro lado se podría mejorar todo lo ya implementado, limando el código ara que fuera más consistente y menos vulnerable, así como dotando de mayor funcionalidad. También es susceptible de mejora todo lo relativo al aspecto gráfico de las interfaces, para lo cual en un ambiente profesional sería necesario la colaboración de al menos un diseñador gráfico.

El comportamiento del NFC y en la estructuración de l aplicación del dispositivo móvil, han sido las dos tareas en las que se ha puesto más ahínco, lo que hace que las mejoras posibles sean de menor calado, aunque todo siempre es digno de ser mejorado. Dejamos estas mejoras propuestas para otro momento o para otra persona, con el deseo de que algún día se complete el desarrollo de la plataforma.

7 Conclusiones

Llegados a este punto de la tesis de máster, se puede afirmar que los objetivos descritos al principio de la misma han sido cumplidos con creces. Durante la elaboración de la tesis se han adquirido gran cantidad de conocimientos en las áreas de interés que motivaron la elección de esta línea de trabajo.

Como ya comentamos anteriormente, el hecho de la ubicuidad y las capacidades computacionales de los dispositivos móviles junto con la gran cantidad de sensores y tecnologías accesorias que disponen, como la tecnología NFC, permiten la creación de nuevas funcionalidades a los usuarios convirtiéndolas en nuevas necesidades, aportando valor añadido sobre modelos de negocio existentes o incluso generando modelos nuevos.

Se ha pretendido poner en práctica la afirmación anterior mediante un caso de aplicación de tecnologías móviles y NFC en el ámbito del canjeo de tickets. Tras su desarrollo, podemos concluir que es posible, a día de hoy, llevar a la práctica de un modo estable toda esta tecnología con soluciones comerciales, lo que indica que son tecnologías de presente.

Sin embargo, las soluciones empresariales que hacen un uso similar de este conglomerado de tecnologías de la información y la comunicación al unísono son realmente escasas, por no decir inexistentes, por lo menos en España. Es evidente que el uso de NFC desde dispositivos móviles esta teniendo una implantación más lenta de lo esperada, lo cual sorprende tras ver la gran cantidad de posibilidades que alberga en muchos ámbitos. Solo nos queda tener paciencia, y esperar el devenir de los acontecimientos bajo el convencimiento absoluto de que la integración tecnológica y la implantación social de la tecnología NFC sea una realidad madura.

Esperemos que dentro de unos años, hablar del éxito social de la tecnología NFC combinada con dispositivos móviles, tales como smartphones, no sea mera ucronía.