E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Gestión de Contratos con Openbravo

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: Aarón Calero Acarreta
Tutor: Alberto Córdoba Izaguirre
Pamplona, 27 de Junio de 2014

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

CONTENIDO

Documento I.

AGRADECIMIENTOS

El trabajo que se expone a continuación no habría podido realizarse sin la ayuda inigualable de OPENBRAVO, Rafa de Miguel, Guillermo Gil y Alberto Córdoba.

RESUMEN

El objetivo de este trabajo es explicar el desarrollo de una aplicación construida sobre Openbravo y más concretamente sobre Openbravo Mobile. Para ello se tomará como ejemplo una aplicación real, y se analizará todo el proceso de desarrollo y gestión, desde las reuniones iniciales con el cliente final hasta la entrega del producto y el periodo de garantía.

Se analizará todo el ámbito del desarrollo del proyecto, pasando tanto por el proceso de desarrollo del código como por la metodología utilizada para gestionar el proyecto. También se explicará qué es y para qué sirve Openbravo, cómo se desarrolla sobre Openbravo y cuales son los requisitos necesarios para poder montar un entorno de desarrollo.

PALABRAS CLAVE

ERP - Openbravo - java - javascript - tecnologías móviles - repositorio - mercurial - eclipse IDE - enyojs - Backbone - Underscore - gestión de contratos - modularidad

1. Introducción

En este trabajo se pretende analizar el proceso de desarrollo de una aplicación de gestión de contratos de publicidad radiofónica para Talent Partners (http://www.talentpartners.com), una empresa neoyorquina cuyo objetivo fundamental es el pago especializado de nóminas.

El trabajo se estructura en cuatro grandes bloques. Este primer bloque se encarga presentar el trabajo y mostrar la estructura del mismo. En el segundo bloque se presentará al cliente. En el tercer bloque hablaremos de qué es Openbravo, y cómo se pueden desarrollar aplicaciones sobre Openbravo y más específicamente sobre Openbravo Mobile. Por último, en el cuarto gran bloque analizaremos varios aspectos de la aplicación construida, incluyendo pero no limitado al origen de la aplicación, la metodología de desarrollo utilizada y los principales procesos o flujos de trabajo presentes en la aplicación.

La idea de la aplicación, que inicialmente iba a llamarse Vita Radio y ha terminado llamándose nAble, nació como extensión de la aplicación VitaTV de Talent Partners, la aplicación original de Talent Partners para gestión de contratos publicitarios principalmente de televisión.

A través de uno de los partners de Openbravo, Transitional Data Services (TDS), Talent Partners contacta con Openbravo para proponer el desarrollo de una aplicación similar a VitaTV, pero construida de manera modular. De este modo se obtendrá una aplicación con una alta usabilidad en un plazo corto, ya que el desarrollo modular agiliza los procesos de diseño y construcción de la aplicación.

Una gran mayoría de los clientes de Talent Partners son agencias de publicidad, y la mayor parte del negocio de anuncios se gestiona en papel. Cada agencia utiliza sus documentos y formularios propios a la hora de contactar con Talent Partners, lo que ralentiza los procesos de gestión. El objetivo del proyecto Vita Radio es estandarizar y digitalizar el proceso de gestión de comerciales de radio, tal y como hizo la aplicación VitaTV con los comerciales de televisión.

Como requisito específico de la aplicación, se pide que pueda ser utilizada en dispositivos móviles (tabletas, smartphones) y que disponga de la capacidad de funcionar en modo offline.

En el apartado 4 se hablará en profundidad de los requisitos de la aplicación y de cómo se repartió el desarrollo de la misma entre Openbravo S.L.U. y otro partner de Openbravo, Fugo Consulting (http://www.fugoconsulting.com/).

2. El Cliente. Talent Partners

Figura 2-1 Logotipo Talent Partners

Talent Partners es una empresa privada norteamericana del ámbito del marketing y la publicidad, con sede en Nueva York, cuyo objetivo es gestionar todos los aspectos relevantes a la hora de lanzar anuncios publicitarios.

Talent Partners nació como hace más de 40 años como una empresa especializada en pagos de nóminas para intérpretes y artistas. Hoy en día, además de procesar más de 65000 nóminas de actores, modelos, músicos, fotógrafos y otra gran variedad de artistas, Talent Partners, se encarga de gestionar todos los procesos concernientes con el lanzamiento de anuncios publicitarios, desde los acuerdos iniciales hasta el lanzamiento del anuncio propiamente dicho.

Hoy en día la empresa mantiene relaciones a largo plazo con más de 700 anunciantes y agencias grandes y pequeñas.

Estos son los productos que ofrece actualmente Talent Partners:

Nóminas

El pago de nóminas especializado es el núcleo sobre el cual se fundó Talent Partners. La empresa es, en el ámbito de la publicidad, una de las mayores pagadoras del mundo, emitiendo más de un millón de nóminas al año alrededor del mundo.

Las soluciones de nóminas especializadas están orientadas a pagos para músicos, modelos, celebridades, fotógrafos, directores, y cualquier otra persona involucrada en la grabación, tanto delante como detrás de la cámara.

Además ofrece soluciones especializadas orientadas a la continuamente creciente área del talento independiente, a ferias, eventos y otros ámbitos.

Asuntos de Negocios

Los asuntos de negocios son servicios de valor añadido que ayudan a hacer frente a necesidades más complejas que aquellas contempladas en los sistemas de pago de nóminas convencionales.

Estos servicios se ofrecen típicamente en función de las necesidades del cliente y van desde negociación de contratos hasta gestión de derechos y patentes.

Global Talent Management

Talent Partners ofrece, a través de este servicio especial, los servicios mencionados anteriormente pero enfocados a un entorno global, donde entrechocan múltiples monedas y legislaturas.

De esta forma TP ofrece, sin costes adicionales excesivos y de manera totalmente transparente, todos sus servicios pero orientados a negocios que quieren prosperar en un entorno sin fronteras.

Otros servicios

Otros servicios que ofrece la empresa incluyen asistencia a la hora de crear campañas publicitarias, asistencia con la distribución de publicidad, un servicio de consultoría para los rodajes, y diferentes cursos y seminarios para aprender a mejorar la calidad de los servicios ofrecidos en el ámbito del marketing y la publicidad.

3. ¿Qué es Openbravo?

Figura 3-1 Logotipo de Openbravo

Openbravo S.L.U. es una compañía de software comercial y open source. Su oficina central está situada en Pamplona, España, y otras oficinas se encuentran ubicadas en Barcelona, México D.F. y Calcuta (la India).

Openbravo fue fundada en 2001, bajo el nombre Tecnicia, por Nicolás Serrano, Ismael Ciordia y Moncho Aguinaga. Fue renombrada a Openbravo en 2006.

Openbravo ofrece dos grandes productos:

- Openbravo ERP, el producto original de la empresa, creado usando como base Compiere ERP.
- Openbravo WebPOS, una aplicación basada en web de tipo punto de venta, llamada LibrePOS antes de ser adquirida por Openbravo.

3.1 El ERP de Openbravo

El producto principal de Openbravo es un software ERP (por sus siglas en inglés: Enterprise Resource Planning, Planificación de Recursos Empresariales).

Un ERP es un software, generalmente un conjunto o suite de aplicaciones, que permite a una empresa u organización recopilar, almacenar, gestionar y procesar información referente a sus actividades de negocio, las cuales incluyen, entre otras:

- Compras y ventas
- Producción
- Logística
- Gestión de inventarios

- Envíos y pagos
- Finanzas
- Gestión de Proyectos

Algunas de las características importantes de las aplicaciones ERP incluyen:

- Diseño modular (cada ámbito de negocio suele ser implementado por un módulo diferente).
- Aplicación configurable: Cada empresa debe poder ajustar la aplicación a sus necesidades. Esto es posible en muchas ocasiones gracias a la modularidad.
- Una base de datos centralizada.
- Los datos son introducidos una única vez en el sistema y desde ahí son reutilizados por todos los componentes. Esto se consigue gracias a la base de datos centralizada.
- Diseño consistente a través de todos los módulos. Aunque la aplicación es modular, al ser utilizada debe sentirse como un todo.

El ERP de Openbravo soporta los siguientes procesos: Compras, ventas, cobros y pagos, gestión de datos maestros (productos, organizaciones, almacenes, terceros, etc.), gestión financiera, gestión de inventarios, gestión de proyectos y servicios, seguridad y acceso basados en roles, y sobre todo una plataforma modular y flexible que soporta la personalización de la aplicación para satisfacer las necesidades de cualquier empresa.

Otro punto importante a destacar del software de Openbravo en general es que trabaja en un entorno de tenencia múltiple, es decir, múltiples clientes pueden convivir en el mismo entorno sin sufrir problemas de privacidad, pues la aplicación aísla cada cliente.

3.2 Contexto tecnológico y estado del arte de las tecnologías Web

Hoy en día las tecnologías Web se encuentran en periodo de transición. Desde su aparición a principios de 1990, la Web ha ido evolucionando hasta convertirse en una parte casi indispensable de nuestro día a día.

El término Web 2.0 fue utilizado por primera vez por Darcy DiNucci en su artículo titulado "Futuro Fragmentado"[1], aunque no fue hasta 5 años más tarde que se hizo popular.

Con la llegada de Web 2.0 los usuarios empezaron a tomar importancia, pasando de ser meros espectadores, unos consumidores pasivos del servicio, a ser unos usuarios activos con cuyas contribuciones pueden alterar el contenido mismo de Internet. Claros ejemplos de este comportamiento pueden apreciarse en el auge de los blogs o la creación de gran cantidad de redes sociales.

Otros ejemplos de la transición a Web 2.0 incluyen las web dinámicas, cuyo contenido varía según las interacciones del usuario, la creación de APIs para permitir el uso automatizado de la información disponible, o la categorización o clasificación de contenido mediante tags o etiquetas, realizada por y para los usuarios de la web.

Sin embargo el término Web 2.0 ha madurado durante los últimos años, y se habla cada vez más de la Web 3.0 y los cambios que acarreará respecto a su predecesora.

El término Web 3.0 es difícil de definir, principalmente porque aún no ha llegado del todo, y porque cada cual tiene una imagen diferente de lo que implicará.

Sin embargo hay tres cambios significativos que están apareciendo cada vez más, y cuyo desarrollo implicaría un avance significativo a la definición de la Web 3.0. Los cambios son los siguientes:

Orientación hacia las tecnologías móviles.

Actividades que, hasta hace pocos años, estaban limitadas a ordenadores de sobremesa o portátiles (como por ejemplo navegar por internet o participar en las redes sociales) son ahora actividades que se realizan continuamente desde tablets, smartphones y otros dispositivos similares.

Esto permite tener acceso a la información en tiempo real y desde cualquier lugar, con la única condición de tener conectividad con la red. Las aplicaciones móviles están cada vez más integradas en nuestro día a día, y nos facilitan múltiples tareas que pueden variar desde hacer la compra hasta consultar el tráfico en la carretera.

Sin embargo este cambio está obligando a la mayoría de las webs a cambiar su formato para ser accesibles y, lo que es más importante, legibles con comodidad desde dispositivos móviles.

Otros desafíos pendientes para las aplicaciones móviles consisten en:

- Encontrar la manera de compatibilizar las aplicaciones con todos, o al menos la mayor cantidad posible, de dispositivos en el mercado, de modo que no sea necesario desarrollar diferentes versiones de la misma aplicación.
- Conseguir aplicaciones que sean capaces de funcionar sin conexión, conservando gran parte de los datos accesibles con

conexión. Esto puede resultar complicado porque la capacidad de almacenamiento de los dispositivos móviles es muy limitada.

Web semántica

En un entorno en el que cada día hay más y más información, se está trabajando para conseguir que las máquinas entiendan la información de un modo similar al de los humanos.

El objetivo de la web semántica es que los buscadores de información no sólo sean capaces de diferenciar contenido apropiado a la búsqueda realizada del contenido incorrecto, sino que además sean capaces de devolver resultados adecuados al ámbito cultural, regional, etc. en el cual se ha realizado la búsqueda.

Las vías de trabajo para lograr este objetivo están abiertas, aunque muchos ven que la solución se encuentra en mejorar los procesos relacionados con la inteligencia artificial, para obtener resultados en la web semántica.

Big Data

El término big data se emplea para representar conjuntos de datos inmensos, del orden que varía desde los petabytes (10^{15} bytes) a los zettabytes (10^{21} bytes), que no pueden ser procesados por los sistemas de bases de datos actuales.

Dado que el tamaño de la web va en aumento constantemente, es cada vez más difícil para los buscadores encontrar información relevante. Conseguir herramientas que permitieran analizar grandes datos permitiría a los navegadores "comprender" el conjunto completo de datos que esconde la web y, de esa forma, obtener resultados mucho más concisos.

3.3. Montando un entorno de desarrollo para Openbravo

En las siguientes secciones se explicarán los pasos a seguir para obtener un entorno de desarrollo para trabajar con Openbravo.

Se hará un análisis general de los requisitos necesarios para montar el entorno, y a continuación se procederá a explicar uno a uno los componentes necesarios para montar el entorno.

3.3.1. Requisitos Generales.

Para montar un entorno de desarollo para trabajar con Openbravo ERP necesitamos las siguientes herramientas:

- Sistema gestor de bases de datos: Oracle o PostgreSQL
- Kit de desarrollo de Java (JDK): Sun JDK u OpenJDK
- Apache Ant
- Apache Tomcat
- Eclipse IDE

Sistema Operativo

Se puede preparar un entorno de desarrollo en múltiples sistemas operativos, aunque se recomienda usar Linux cualquier otro sistema operativo UNIX (BSD, OS-X, etc.) pues el número de herramientas disponibles para desarrolladores es mayor que en otros sistemas.

Sin embargo es preferible utilizar un sistema operativo con el que nos sintamos confortables a la hora de trabajar. Está demostrado que se obtienen peores resultados si se trabaja con un sistema operativo desconocido o que se le hace ajeno al desarrollador.

En cualquier caso, se requiere crear un usuario sin privilegios para desarrollar. Una vez el entorno está configurado, no se debe ejecutar ningún comando con permisos de superusuario/root (en sistemas UNIX) o administrador (en sistemas Windows) para nada que tenga que ver con el desarrollo de openbravo, especialmente tareas ant para compilar el sistema.

Se requiere que el sistema tenga por lo menos 2 Gigabytes de memoria RAM en arquitecturas de 32 bits y 3 Gigabytes en arquitecturas de 64 bits. Para una experiencia de desarrollo fluida es aconsejable disponer de 8 Gigabytes de memoria RAM.

En lo que sigue se considerará una máquina Linux (Ubuntu 13.04) con PostgreSQL como gestor de bases de datos y OpenJDK como kit de desarrollo de java.

3.3.2. Gestor de bases de datos: PostgreSQL

Figura 3-2 Logotipo de PostgreSQL

PostgreSQL es un potente gestor de bases de datos relacionales de código abierto. Es compatible con la mayoría de los sistemas operativos importantes en la actualidad, incluyendo Linux, UNIX (BSD, Solaris, Mac OS X, ...) y Windows.

Es completamente compatible con las propiedades ACID (Atomicity, Consistency, Isolation and Durability - Atomicidad, Consistencia, Aislamiento y Durabilidad) que aseguran que las operaciones en la base de datos puedan ser consideradas transacciones.

Openbravo ERP soporta únicamente las versiones de PostgreSQL entre 8.3.5 y 9.1.

Tomaremos por ejemplo la instalación de la versión 9.1 de PostgreSQL.

Para proceder a su instalación abriremos una terminal de comandos con permisos de superusuario y ejecutaremos lo siguiente:

```
$> apt-get install postgresql-9.1
$> apt-get install postgresql-contrib
$> apt-get install pgadmin3
```

El primer comando instalará el sistema gestor de bases de datos en nuestro sistema. Esto creará un usuario en el sistema llamado postgres.

El segundo comando instalará el paquete UUID-contrib requerido por Openbravo ERP.

Por último el tercer comando instalará el interfaz gráfico para acceder y trabajar con las conexiones a las bases de datos.

Una vez instalados los paquetes requeridos es necesario realizar diversos cambios en la configuración de PostgreSQL.

Para empezar editamos el fichero de configuración postgresql.conf (su ubicación depende del sistema en el cual se ha instalado PostgreSQL) y añadimos la línea

```
listen_addresses = '*'
```

A continuación editamos el fichero pg_hba.conf y añadimos la línea

```
host all all 0.0.0.0/0 md5
```

También es necesario modificar la contraseña del usuario postgres, tanto a nivel de sistema como a nivel de base de datos. Para ello primero modificamos la contraseña en la base de datos:

```
$> sudo su postgres -c psql template1
# ALTER USER postgres WITH PASSWORD 'postgres'
#\q
A continuación editamos la contraseña del usuario en el sistema:
$> sudo passwd -d postgres
$> sudo su postgres -c passwd
```

Introducimos y confirmamos la contraseña 'postgres'

Por último debemos reiniciar el servicio postgresql para que los cambios realizados entren en acción. Para ello tenemos dos opciones (ambas tienen que ser ejecutadas con permisos de usuario):

```
$> /etc/init.d/postgresql restart
```

ó

```
$> service postgresql restart
```

3.3.3. Kit de Desarrollo de Java (JDK).

Procedemos a instalar OpenJDK version 6 del mismo modo que hemos procedido a instalar PostgreSQL previamente, mediante una consola de comandos con permisos de superusuario. Ejecutamos el comando:

```
$> apt-get install openjdk-6-jdk
```

A continuación es necesario añadir, si no está ya, la variable $JAVA_HOME a las variables de entorno y añadir tambien la ruta $JAVA_HOME/bin a la variable $PATH.

Se pueden realizar fácilmente ambas acciones editando o creando el fichero ~/.bashrc y añadiendo las líneas:

```
export JAVA_HOME=<ruta a la instalación de java>
export PATH=$PATH:$JAVA_HOME/bin/
```

Para probar que el JDK se ha instalado correctamente podemos ejecutar los siguientes comandos en un nuevo terminal:

```
$> echo $JAVA_HOME
$> java -version
```

3.3.4 Apache Ant

Figura 3-3 Logotipo de Ant

Apache Ant es una librería java y una herramienta de línea de comandos completamente multiplataforma cuyo objetivo principal es automatizar el proceso de construcción de proyectos basados en java.

Openbravo utiliza Ant no sólo para compilar el proyecto java, sino también para crear y poblar la base de datos de la aplicación, descargar los ficheros necesarios para la configuración, o desplegar la aplicación a un servidor web, por ejemplo.

Para instalar Apache Ant ejecutamos desde un terminal de comandos con permisos de superusuario el comando:

```
$> apt-get install ant
```

Y a continuación añadimos las variables de entorno $ANT_HOME y $ANT_OPTS y modificamos la variable $PATH añadiendo las siguientes líneas al fichero ~/.bashrc:

```
export ANT_HOME=<ruta a la instalacion de Ant>
export ANT_OPTS="-Xmx1024M -XX:MaxPermSize=128M"
export PATH=$ANT_HOME/bin/:$PATH
```

3.3.5 Apache Tomcat

Figura 3-4 Logotipo de Tomcat

Apache Tomcat es el servidor web utilizado por Openbravo para servir la aplicación.

Al contrario que en un entorno de producción, se desaconseja utilizar el Tomcat proporcionado por el gestor de paquetes del sistema (por ejemplo el Tomcat que se instala mediante apt-get). En su lugar, se recomienda descargar la distribución oficial de Apache Tomcat y descomprimirla a ~/servers/tomcat/.

Openbravo no está probado sobre Apache Tomcat 7, por lo que está recomendado usar la versión 6.0.x

Es necesario añadir a las variables de entorno las variables $CATALINA_HOME, $CATALINA_BASE y $CATALINA_OPTS y modificar la variable $PATH. Como con las aplicaciones anteriores podemos realizar estas modificaciones editando el fichero ~/.bashrc y añadiendo las líneas:

```
export CATALINA_HOME=~/servers/tomcat/
export CATALINA_BASE=~/servers/tomcat/
export CATALINA_OPTS="-Djava.awt.headless=true -Xms384M -Xmx512M -XX:MaxPermSize=256M"
export PATH=$CATALINA_HOME/bin/:$PATH
```

3.3.6. Software de Control de Versiones: Mercurial

Como se explicará en un apartado posterior, Openbravo utiliza un software de control de versiones para mantener un control sobre los ficheros que conforman la aplicación.

Mercurial es un software de control de versiones distribuido e independiente de la plataforma utilizada. Su funcionalidad puede ser extendida a través de paquetes de extensión tanto oficiales (incluidos en la propia distribución de Mercurial) como desarrollados por contribuidores, y es

completamente open source licenciado bajo la GNU General Public License Version 2.

Los repositorios oficiales de Openbravo (code.openbravo.com) están gestionados mediante Mercurial por lo que es necesario instalar el software para poder descargar los módulos necesarios para desarrollar.

Podemos instalar Mercurial desde línea de comandos mediante el comando:

```
$> apt-get install mercurial
```

Se recomienda la creación del fichero ~/.hgrc para especificar un nombre de usuario para realizar ciertas operaciones en los repositorios. Este fichero también se utiliza para habilitar puntos de extensión de mercurial, por ejemplo programas para realizar comparaciones (diffs) entre ficheros.

3.3.7. Entorno de Usuario: Eclipse IDE e instalación de los ficheros fuente de Openbravo.

Una vez tenemos instalados y configurados todos los componentes necesarios para desarrollar con Openbravo, es necesario preparar el entorno de desarrollo y descargar todos los ficheros fuente que forman la aplicación Openbravo.

Para una óptima experiencia de desarrollo, Openbravo recomienda trabajar con la versión Indigo del entorno de desarrollo integrado (IDE) Eclipse.

Una vez tenemos Eclipse instalado es necesario instalar el código fuente desde el repositorio externo. Se considera una buena práctica mantener dos copias locales del repositorio externo; una copia original o limpia del repositorio, la cual mantendremos constantemente actualizada, y el entorno de trabajo sobre el cual realizaremos nuestros desarrollos.

También se aconseja mantener los repositorios fuera de la ruta de instalación de Eclipse.

Para proceder a la descarga del código fuente utilizaremos Mercurial para copiar el repositorio externo a una carpeta local. Tenemos dos opciones, copiar la última versión estable de la aplicación, o copiar el entorno de pre-integración (pi), con todos los desarrollos realizados hasta la fecha. Se recomienda a los desarrolladores trabajar con el repositorio pi.

Para copiar el repositorio a nuestra máquina ejecutaremos el siguiente comando:

```
hg clone https://code.openbravo.com/erp/devel/pi
```

el cual nos creará la carpeta pi en el directorio actual.

Cambiamos al directorio pi para proceder a la configuración del entorno.

Primero ejecutamos

```
ant setup
```

que descargará al subdirectorio config un ejecutable de acuerdo a nuestro sistema operativo. Dicho ejecutable lanza un interfaz gráfico que ayuda a configurar diversos parámetros de la aplicación, como por ejemplo la base de datos. A la hora de definir las credenciales del administrador de la base de datos es necesario utilizar los valores utilizados en la configuración del SGBD (apartado 3.2.2)

Cambiamos al subdirectorio config y ejecutamos el fichero recién descargado setup-properties-[NOMBRE_SO].[EXT]. Tras finalizar el asistente de configuración se creará un fichero llamado Openbravo.properties con la configuración establecida. La aplicación ya está lista para ser instalada.

A continuación volvemos al directorio pi y ejecutamos el comando

```
ant install.source
```

el cual:

- Crea y puebla la base de datos con unos valores por defecto.
- Genera los ficheros fuente
- Y los compila para que puedan ser posteriormente desplegados a un servidor web.

Este proceso puede tardar hasta 25 minutos, por lo que se aconseja redirigir la salida del mismo a un fichero de texto. Para poder redirigir la salida a un fichero de texto y al mismo tiempo ver la salida por consola puede utilizarse *tee* en Linux o *mtee* en Windows.

Una vez termina el proceso de instalación de los ficheros fuente procedemos a importar los proyectos en Eclipse, crear el servidor y lanzar la aplicación.

Nota: Se aconseja deshabilitar la opción *Build Automatically* en Eclipse.

Se deben importar cuatro proyectos:

- Openbravo
- Openbravo Core (subcarpeta src-core)

- Openbravo Trl (subcarpeta src-trl)
- Openbravo Wad (subcarpeta src-wad)

A continuación creamos el servidor Tomcat, indicando como ruta de instalación ~/servers/tomcat si se ha seguido la ruta propuesta en el apartado 3.2.5. Mientras se completa el asistente de creación es necesario añadir Openbravo a la lista de recursos configurados (configured resources) por el servidor.

Una vez creado el servidor es necesario realizar unos ligeros ajustes a su configuración. Para ello hacemos doble click en la instancia creada del servidor y realizamos los siguientes cambios:

- En el apartado Server Options marcamos el check Serve Modules without publishing
- En Timeouts asignamos 120 segundos a ambos timeouts. Estos tiempos pueden modificarse si nuestra aplicación requiere más tiempo para iniciarse.
- Entramos en Open Launch Configuration y en la pestaña Arguments añadimos la siguiente línea al final: **`-server -Djava.awt.headless=true -Xms384M -Xmx1536M -XX:MaxPermSize=256M`**

Guardamos los cambios realizados pulsando Ctrl+s.

Por último es necesario importar ciertas preferencias preconfiguradas para el entorno. El fichero con dichas preferencias se encuentra en la subcarpeta config/eclipse/.

Una vez terminado el proceso de configuración podemos reactivar la opción Build Automatically, seleccionar los cuatro proyectos, y refrescarlos pulsando la tecla F5.

Ponemos en marcha el servidor Tomcat y, si todo ha sido configurado correctamente, en la dirección `http//localhost:8080/openbravo` nos encontraremos la página de login del ERP, en la cual se pueden utilizar las siguientes credenciales para conectarse:

- Usuario: Openbravo
- contraseña: openbravo

En este punto ya disponemos de un entorno listo para empezar a desarrollar.

3.4. Arquitectura de Openbravo 3

Openbravo 3.0 pasa de la arquitectura basada en servlets utilizada por la versión 2.5 a una arquitectura RIA (rich internet applications, en castellano aplicaciones de Internet enriquecidas). Esto permite optimizar las peticiones al servidor, separando por un lado la carga del interfaz de usuario, que se realiza una única vez, y por otro los datos de la aplicación, que se piden al servidor cada vez que son requeridos.

La imagen siguiente representa los módulos que componen la arquitectura de Openbravo 3.0

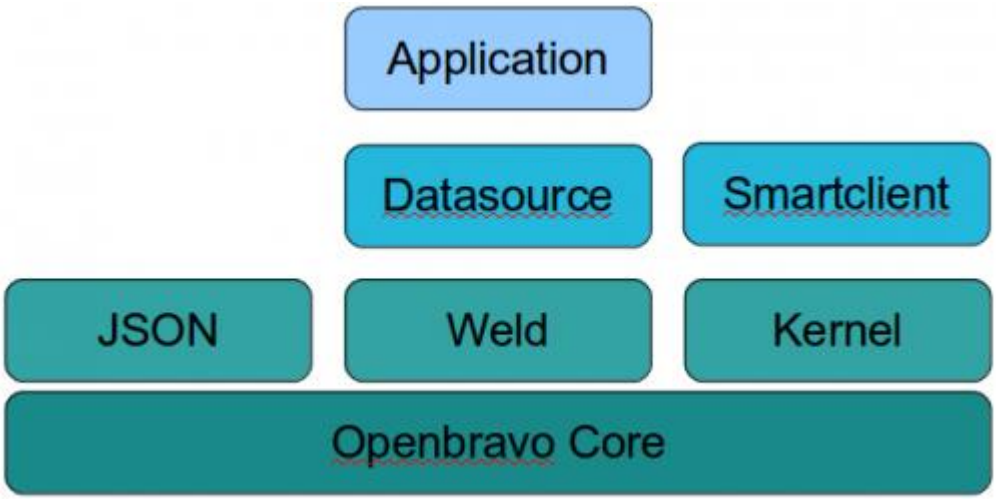

Figura 3-5 Estructura de Openbravo 3.0

El núcleo de Openbravo, Openbravo Core, es el módulo que incluye todas las funcionalidades básicas del ERP.

El módulo JSON proporciona el servicio web JSON REST, utilizado en el intercambio de datos entre el cliente y el servidor.

El módulo Weld proporciona capacidad de injección de dependencias y gestión de componentes. Weld es una de las implementaciones más utilizadas del estándar CDI (Contexts and Dependency Injection for the Java EE Platform), que define mecanismos para inyección de dependencias, ciclos de vida de los objetos dependientes del contexto, soporte a la estructura modular y otra serie de servicios orientados a mejorar la estructura del código de aplicación.

El módulo Kernel se encarga de varias tareas de la infraestructura de Openbravo, por ejemplo procesamiento de peticiones, manejo de eventos y compresión y cacheado de datos.

El módulo Datasource hace uso del módulo JSON para proporcionar una implementación más concreta y de más alto nivel de las funcionalidades de consulta y recuperación de datos y otras tareas relacionadas.

El módulo Smartclient es el marco de trabajo utilizado por Openbravo ERP para crear y manejar los componentes del interfaz de usuario.

Por último el módulo de aplicación contiene la implementación de los formularios, barras de navegación y tablas de datos así como código de la aplicación, tanto del lado del servidor como del cliente.

Como se puede apreciar en la siguiente imagen, Openbravo incluye una serie de módulos adicionales que proporcionan nuevas funcionalidades (por ejemplo Widgets) o integraciones con servicios externos (por ejemplo integración con el API de Google o con el protocolo de autentificación OpenID).

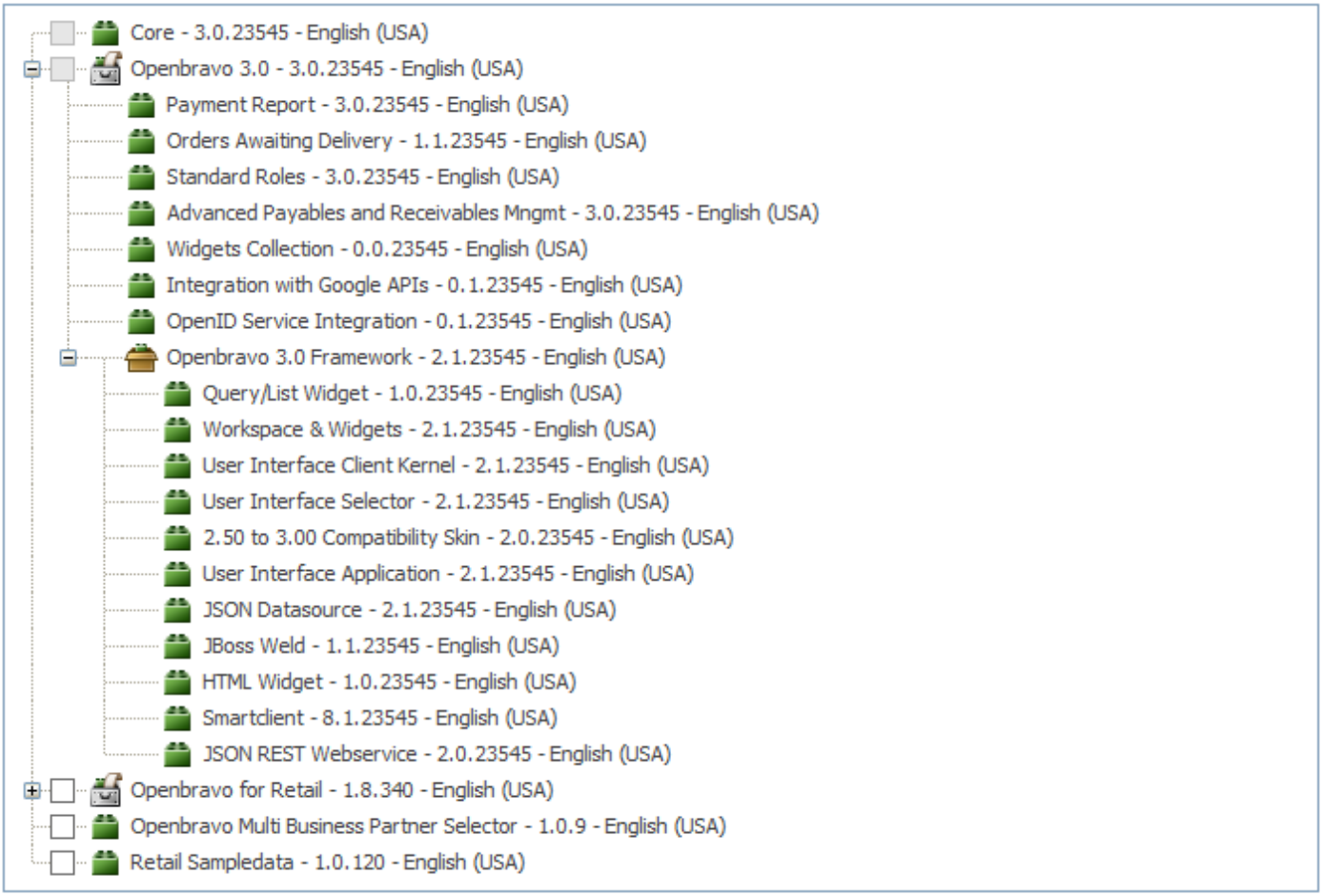

Figura 3-6 Vista de la consola de gestión de módulos de una instalación de Openbravo 3.0

3.5 Openbravo Mobile

Openbravo Mobile es un módulo de Openbravo que extiende la plataforma Openbravo con capacidades para el interfaz de usuario basadas en HTML-5.

Ofrece una infraestructura que permite el desarrollo de aplicaciones web móviles sobre Openbravo. Dichas aplicaciones están optimizadas para dispositivos móviles y pantallas táctiles, y pueden ser utilizadas en modo desconexión.

3.5.1 Arquitectura de Openbravo Mobile

La siguiente imagen muestra los componentes que conforman la arquitectura de Openbravo Mobile:

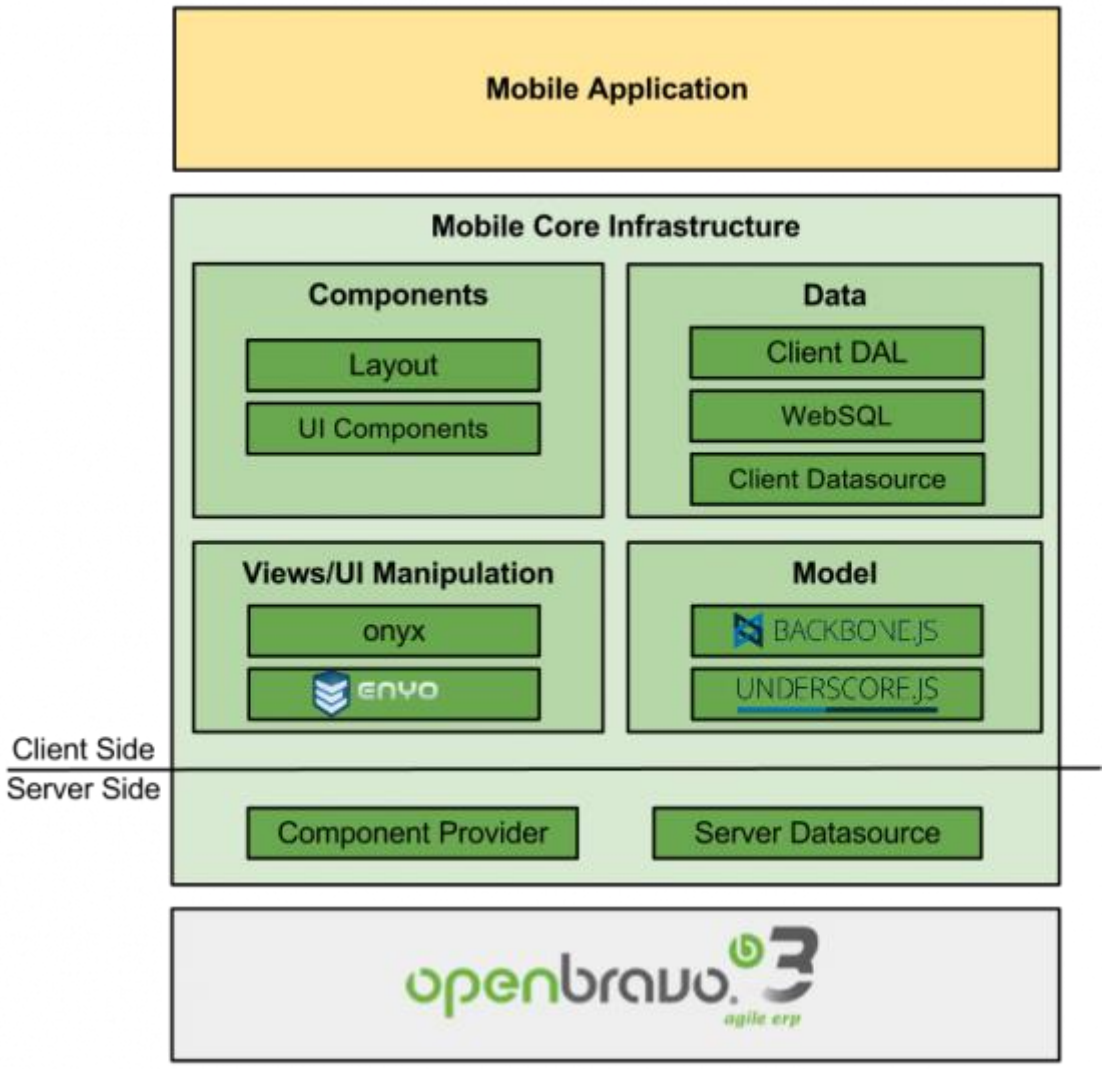

Figura 3-7 Estructura de Openbravo Mobile

La arquitectura en el lado del servidor está formada por el *component provider* o proveedor de componentes, y por el *server datasource* o fuente de datos del servidor.

El proveedor de componentes se encarga de informar al sistema de los recursos estáticos (ficheros javascript, ficheros css, imágenes, etc.) que se incluyen en la aplicación.

La fuente de datos es el componente que se encarga de servir los datos de aplicación al lado del cliente (por ejemplo datos almacenados en una base de datos) y se encarga también de procesar todos aquellos datos capturados en la aplicación cliente que serán incluidos en el ERP.

Los componentes que forman la arquitectura del lado del cliente pueden ser divididos en cuatro grandes grupos de acuerdo a su propósito: datos, componentes, vista y manipulación del interfaz y modelo de datos.

Los componentes referentes a datos incluyen el Client DAL (Client Data Access Layer - Capa de acceso a datos del cliente), WebSQL y el Client Datasource o fuente de datos del cliente.

WebSQL es la tecnología que permite almacenar datos en el lado del cliente (por ejemplo creando una base de datos en la caché del navegador) para poder acceder rápidamente y con facilidad. Esta base de datos es poblada gracias a los datos recibidos desde las fuentes de datos de servidor y cliente.

La Capa de Acceso a Datos es uno de los componentes más importantes del ERP de Openbravo. Provee un interfaz para acceder de manera sencilla a los datos almacenados en la base de datos, ya sea en la WebSQL o en la base de datos del servidor.

La fuente de datos del cliente es el extremo opuesto a la fuente de datos del servidor, y su función consiste en recibir y procesar datos del servidor, así como enviar de vuelta aquellos datos que tienen que ser guardados en el ERP.

Los componentes referentes al modelo incluyen las librerías javascript Underscore y Backbone.

Underscore proporciona una serie de métodos para trabajar con colecciones de datos, mientras que Backbone es un marco de trabajo que implementa el paradigma MVC (modelo-vista-controlador) en javascript. En concreto Openbravo Mobile utiliza Backbone para la representación del modelo de datos.

Los componentes utilizados para representar las vistas y controlar la manipulación de elementos del interfaz de usuario son otras dos librerías de javascript, enyojs y su extensión, onix.

Enyo es el framework que utiliza Openbravo Mobile para crear los componentes que conforman todas las ventanas de la aplicación móvil. Además incluye un potente motor de manejo de eventos y permite la sobreescritura de componentes para ajustarlos a cualquier necesidad.

Onix es un conjunto de herramientas que extiende de Enyo y contiene multitud de componentes de interfaz ya configurados, como por ejemplo botones o pop-ups.

Los componentes de la arquitectura como tal incluyen el layout o disposición de los elementos en pantalla así como los propios componentes del interfaz de usuario.

Gracias a los paneles predefinidos de onix, Openbravo Mobile utiliza una disposición multicolumna. Dependiendo de la resolución de pantalla del dispositivo utilizado la aplicación muestra dos columnas o una única columna con posibilidad de cambiar a la segunda deslizando el dedo sobre la pantalla (sólo en dispositivos con pantalla táctil; también existe un botón que permite cambiar entre columnas).

Por último todos los componentes del interfaz de usuario, ya sean botones, menús, teclados virtuales, pop-ups, etcétera son creados utilizando componentes de enyo, llamados kinds (en castellano, clases o tipos).

Veamos ahora más en detalle algunos de los componentes mencionados.

3.5.2. EnyoJS

Figura 3-8 Logotipo de EnyoJS

EnyoJS es un framework javascript orientado a objetos y multiplataforma, disponible bajo licencia Apache 2.0.

Historia

Enyo 1 nació como un framework creado por HP para desarrollar aplicaciones para su TouchPad, un dispositivo que corría sobre webOS, y eligió el entorno web como su entorno de aplicación primario. Todas las aplicaciones que incluía el TouchPad, y muchas más, fuero desarrolladas haciendo uso de Enyo I.

A pesar de que se pensó como un framework multiplataforma, Enyo I se orientó principalmente y por necesidad a webOS, y más concretamente a TouchPad.

En enero de 2012 HP liberó Enyo bajo la licencia Apache 2.0, y dirigió sus esfuerzos a convertir Enyo en un framework verdaderamente plataforma, incluyendo todos los entornos web actuales (iOS, Android, Safari, Firefox, Chrome e IE8+). En julio de 2012 Enyo 2.0 terminó su versión beta y fue lanzada al público.

En marzo de 2013 LG Electronics se hizo con los grupos de trabajo tanto de Enyo como de webOS, y extendió el objetivo multiplataforma de Enyo, haciéndolo compatible con aplicaciones para Smart TV.

Recientemente, en febrero de 2014, el equipo de Enyo ha liberado las librerías Moonstone y Spotlight, las cuales proveen un conjunto de herramientas para optimizar aplicaciones en monitores amplios.

Conceptos Clave

Enyo consiste en un modelo simple pero potente de encapsulación, el cual ayuda a factorizar funcionalidades en bloques autocontenidos que son fáciles de reusar y mantener.

Todos y cada uno de los elementos en una aplicación Enyo son **componentes**. Los componentes, a su vez, están compuestos por otros componentes. Por ejemplo, es fácil combinar dos etiquetas <*label*> y <*input*> en un componente llamado *LabeledInput.* Cualquier componente, por complejo que sea (un selector de fechas, una calculadora, etc.) puede ser empaquetado como un componente reusable.

Utilizando Enyo se puede aplicar el dicho “divide y vencerás” a proyectos grandes. Dado que Enyo se basa en componentes, y estos componentes pueden combinarse para crear componentes aún mayores, aplicaciones complejas pueden estar compuestas por muchos componentes pequeños que, en conjunto, realizan la misma función.

Gracias a la modularidad de Enyo es posible reutilizar componentes en un mismo proyecto, en diferentes proyectos o incluso empaquetarlos para compartirlos con otros desarrolladores.

Estructura

Enyo está estructurado en diferentes partes, cada una de ellas ofreciendo diferentes funcionalidades. Básicamente una distribución de Enyo se compone de lo siguiente:

- **enyo -** el núcleo de la aplicación
- **lib** - carpeta para plugins y librerías
  - **onyx**
  - **layout**
  - **canvas**
  - **extra**

- **enyo**

Es el núcleo de la aplicación, y proporciona características que permiten construir aplicaciones complejas. Dichas características se presentan de manera modular, y podrían llegar a separarse de ser necesario, pero han sido empaquetadas de manera conjunta para ofrecer un conjunto de utilidades completo.

Las características que conforman el núcleo de Enyo son las siguientes:

- Encapsulación de objetos, herencia y jerarquías de propiedad, obtenidas mediante componentes y “clases” o kinds.
- Vistas y modelos de objetos del DOM – UiComponent y Control
- Enrutamiento de eventos: Dispatcher y Signals
- Async and Ajax
- Componentes básicos HTML - Base UI
- Soporte táctil - Touch scroller y simulador de gestos
- Cargador de paquetes - enyo.depends() y package.js

- **onyx**

La librería onyx proporciona un conjunto de herramientas para construir componentes del interfaz de usuario (botones, campos de entrada, barras de progreso, etc.) para aplicaciones de escritorio y aplicaciones móviles.

- **layout**

Esta librería puede usarse para maquetar los componentes que conforman la aplicación, para no depender de técnicas puras de HTML y CSS. Los componentes que ofrece esta librería son compatibles con todos los navegadores soportados.

- **Canvas**

Canvas es una librería orientada al interfaz de usuario, y su objetivo principal es mostrar cómo se encapsulan elementos HTML en objetos del tipo enyo.Control.

La librería es meramente demostrativa y no pretende ser un conjunto de herramientas completo para el manejo de elementos del lienzo de la página web.

- **extra**

La librería **extra** incluye diversos componentes que son utilizados en numerosos ejemplos y demos.

3.5.3. Underscore.js

Underscore, _ o Underscore.js es una librería javascript que contiene un conjunto de funciones que proporcionan un gran rango de funcionalidades, desde los clásicos mapeos o llamadas a funciones hasta manejo de plantillas, tests de igualdad, y más.

Siempre que sea posible Underscore utiliza las funciones nativas de los navegadores.

Las funciones que ofrece Underscore se clasifican en los siguientes apartados:

- Collections (Colecciones de objetos)
- Arrays
- Functions (Funciones de… sí, funciones :P)
- Objects (Objetos)
- Utilities (Utilidades)
- Chaining (Encadenamiento)

Underscore.js se encuentra actualmente en la versión 1.6.0.

3.5.4. Backbone.js

Figura 3-9 Logotipo de Backbone

Backbone (literalmente, columna vertebral) es un framework javascript que permite estructurar el código de las aplicaciones utilizando el paradigma MVC (modelo, vista, controlador). Para ello proporciona **modelos** con atributos del tipo clave-valor y eventos personalizados, **colecciones de objetos** con una API completa de funciones de enumeración, y **vistas** con manejo de eventos declarativo. Backbone conecta estos tres pilares mediante un interfaz JSON RESTful.

Backbone depende de Underscore.js. Además, para proporcionar funcionalidades de persistencia REST, soporte de históricos vía Backbone.router y manipulación del DOM con Backbone.View, también es necesario incluir jQuery (y json2.js para poder dar soporte a versiones de Internet Explorer antiguas).

Backbone vio la luz el 13 de Octubre de 2010, en la versión 0.1.0, y hoy en día se encuentra ya en la versión 1.1.2, desplegada el 20 de febrero de 2014.

Introducción

Cuando se desarrollan aplicaciones web que implican mucho código javascript, es común acabar con montones enredados de selectores jQuery y llamadas a funciones, todos intentando mantener en sincronía los datos entre el HTML, el interfaz, y la base de datos. Para este tipo de aplicaciones, es mejor aplicar un enfoque más organizado o estructurado.

Con Backbone los datos se representan mediante modelos, y dichos modelos pueden ser creados, validados, destruidos o almacenados en el servidor.

Cada vez que una acción en el interfaz de usuario provoca un cambio en algún atributo de un modelo, éste lanza automáticamente un evento de cambio (*change event*). Todas las vistas que incluyen el modelo mencionado pueden recibir el evento y responder al mismo, por ejemplo recargándose para mostrar la información actualizada. En una aplicación que utiliza Backbone , no es necesario programar el código que sincroniza los datos con la vista; cuando el modelo cambia, las vistas se actualizan automáticamente.

La filosofía de Backbone es proporcionar el conjunto mínimo necesario de funciones primitivas para construir aplicaciones web utilizando javascript. En un entorno donde los frameworks generalistas son muy comunes, y las librerías de funciones por lo general requieren reestructurar las aplicaciones para poder integrarse, Backbone pretende dar libertad para diseñar aplicaciones permitiendo al mismo tiempo explotar todas las posibilidades de las funcionalidades web.

Eventos (Backbone.Events)

El módulo Events es un módulo que puede adjuntarse a cualquier objeto, proporcionándole la capacidad de enlazar eventos personalizados, así como dispararlos para que otros objetos puedan recibirlos.

Estos evento personalizados no tienen por qué ser declarados previamente, y pueden incluir argumentos que se enviarán junto con el evento.

Backbone.Events incluye un catálogo de eventos predefinidos que se disparan bajo ciertas condiciones (por ejemplo el evento **add** cuando se añade un modelo a una colección, el previamente mencionado **change** cada vez que cambia un atributo de un modelo, e incluso un evento especial **all** que se dispara para todos los eventos, e incluye como primer parámetro el nombre del evento ocurrido).

Por lo general, y aunque no sea recomendable, Backbone permite evitar lanzar eventos pasando la opción {silent: true} a la función que se quiere silenciar.

Modelos (Backbone.Model)

Los modelos son el núcleo de cualquier aplicación javascript. Contienen todos los datos, así como gran parte de la lógica de la aplicación (validaciones, propiedades calculadas, control de acceso).

Para usar Backbone.Model se crea un objeto haciendo uso de extend() (var myModel = Backbone.Model.extend({}); crearía un modelo vacío, sin funciones ni atributos propios pero con todos los métodos de Backbone.Model). En términos de programación orientada a objetos extend() puede entenderse como el proceso de herencia, y cada modelo puede entenderse como una clase. Dicho esto, se observa que podemos hacer extend() sobre un modelo ya creado para crear modelos más específicos.

Backbone.Model incluye una gran variedad de métodos para trabajar con los modelos, entre otros **get** y **set** para cambiar/examinar atributos de un modelo, **fetch** y **save** para recuperar y guardar datos en el servidor, y también métodos de validación de modelos.

Colecciones (Backbone.Collection)

Las colecciones son conjuntos ordenados de objetos. Entre otras cosas, se pueden enlazar eventos **change** para notificar cambios en cualquiera de los modelos que conforman la colección, escuchar eventos **add** y **remove** para controlar los modelos que se añaden o eliminan de la colección, utilizar la función **fetch** para recuperar la colección del servidor, y, como ya se ha comentado previamente, utilizar un conjunto de funciones de Underscore para manejar conjuntos de datos.

Router (Backbone.Router)

Las aplicaciones web, por lo general, ofrecen URLs compartibles, que pueden enlazarse y, por ejemplo, añadirse a favoritos/marcadores. Hasta hace poco se utilizaban fragmentos hash (#page) para proporcionar estos "permalinks" o links permanentes. Con la llegada de la History API (http://diveintohtml5.info/history.html) es posible utilizar URLs estándar (/page) para representar los mismos permalinks.

Backbone.Router proporciona métodos para enrutar páginas de la aplicación y conectarlas a acciones y eventos. Para navegadores que no soportan la History API, Router convierte automáticamente las URLs para que sean compatibles.

History (Backbone.History)

Backbone.History funciona como el router global de toda la aplicación, y maneja de manera transparente los eventos *hashchange* y la transformación automática de permalinks a URLs estándar y viceversa (dependiendo de las capacidades del navegador utilizado).

Incluye el método start() para indicarle a la aplicación que puede empezar a escuchar eventos *hashchange*.

View (Backbone.View)

Las vistas de Backbone son una utilidad para organizar el interfaz de usuario. Muchas vistas van enlazadas con un modelo, de modo que cada vez que se lanza un evento de cambio en ese modelo, se ejecuta la función render() de la vista, redibujándola en la aplicación. De este modo, allá donde esté representado ese modelo, sus datos se encontrarán actualizados a tiempo real.

Utilidades

Backbone incluye el método Backbone.noConflict() y el atributo Backbone.$. El método noConflict permite devolver el objeto Backbone a su estado original, permitiendo insertar Backbone en páginas ajenas, sin aplastar el Backbone existente. El atributo $, en cambio, permite especificar qué librería utilizar para manejar Ajax y el DOM.

En concreto Openbravo Mobile utiliza Backbone.js para representar datos, es decir, hace uso de Backbone.Model y Backbone.Collection.

3.5.5. WebSQL

WebSQL es una API (Application Programming Interface - Interfaz de programación de aplicaciones) web para almacenar bases de datos locales que pueden ser consultadas usando un dialecto de SQL. El SGBD (sistema gestor de bases de datos) utilizado por WebSQL es SQLite.

WebSQL está soportado por los siguientes navegadores: Google Chrome, Opera, Safari y AndroidBrowser.

Desde noviembre de 2010 el mantenimiento de WebSQL ha quedado descontinuado, principalmente a causa de no soportar implementaciones sobre diferentes SGBDs, y la organización W3C recomienda utilizar Web Storage o IndexedDB.

Openbravo utiliza WebSQL para almacenar en el navegador los datos recuperados desde el servidor gracias a Datasource. De esta forma la carga de datos se realiza una única vez, al cargar la aplicación, y los accesos siguientes pueden hacerse rápidamente y con facilidad.

El mayor problema que surge con el uso de datos almacenados localmente es la sincronización. ¿Qué ocurre cuando el usuario modifica datos desde la aplicación? ¿Cómo tienen que sincronizarse esos datos con los almacenados en el servidor? ¿Que ocurre si, por casualidad, se modifican los mismos datos en el servidor y en el cliente? ¿Cual de las dos versiones es la que debería perdurar)

Se dedicará una sección posterior a explicar en profundidad el proceso de sincronización de los datos en VitaRadio (nAble).

4. La Aplicación

4.1. Trasfondo de la aplicación

Como se ha comentado previamente, Openbravo trabaja a través de partners en prácticamente todas las ocasiones. En este caso, el cliente Talent Partnes contactó con TDS, Transitional Data Services, para desarrollar una aplicación cuyo objetivo es gestionar el ciclo de los contratos para intérpretes de anuncios de radio.

4.1.1. De VitaTV a VitaRadio y de VitaRadio a nAble

Previamente al desarrollo de esta aplicación, Talent Partners ya poseía una aplicación para gestionar contratos de anuncios televisivos, llamada VitaTV. Partiendo del diseño y funcionamiento de esa aplicación se pidió el desarrollo de VitaRadio, pues la funcionalidad básica sería similar.

Además el desarrollo requería utilizar diversos procesos de VitaTV como puente con otros sistemas de Talent Partners, por ejemplo ordenador central de la empresa o el gestor de documentos.

En esta sección veremos los requisitos (refiriéndonos por requisitos a los requisitos para el desarrollo y no los requisitos de la aplicación) y el alcance funcional propuesto por TDS a Openbravo para el desarrollo de la aplicación.

4.1.1.1. Requisitos

El objetivo de VitaRadio, como ya se ha comentado, es gestionar el ciclo de contrato para performers o intérpretes de anuncios radiofónicos.

Talent Partners quiere conseguir una aplicación diseñada de manera modular, con todas las funcionalidades importantes disponibles por defecto, en lugar de tener una aplicación completamente personalizada y dificil de extender.

Se escoge Openbravo ERP como base de la aplicación para cumplir los siguientes objetivos simultáneamente:

- Comercializar VitaRadio en el menor tiempo posible gracias a la agilidad de desarrollo modular.
- Conseguir un interfaz de usuario altamente usable y móvil, con una experiencia de usuario "Business to Customer" o del negocio al

consumidor, para los procesos realizados por los productores en la calle.

- Proporcionar una capa de aplicación poco acoplada (modular) y altamente productiva para despliegue y mantenimiento de futuras aplicaciones.

4.1.1.2. Alcance Funcional

La mayor parte del negocio de la publicidad radiofónica que gestiona Talent Parners se realiza a papel, y cada agencia utiliza sus propios formularios para estar en contacto con Talent Partners.

El objetivo de VitaRadio es estandarizar y digitalizar el proceso de gestión de anuncios de radio, tal como hace VitaTV con en el ámbito de la publicidad televisiva.

Muchos de los clientes de Talent Partners son agencias, las cuales gestionan el proceso creativo y las sesiones de grabación de los anuncios de radio. Normalmente dichas sesiones se realizan en un estudio de grabación, peor es necesario considerar que, gracias a las tecnologías móviles, es posible desarrollar las grabaciones en virtualmente cualquier escenario

La mayoría de las sesiones incluyen a un solo intérprete; algunas tienen dos y es posible, aunque ocurre en muy pocas ocasiones, que el grupo intérprete sea mayor.

A un alto nivel de abstracción, se requiere que la aplicación cumpla los siguientes tres procesos:

- Crear la sesión en la aplicación.
- Añadir información de los participantes en la grabación. La aplicación requiere un contrato por cada intérprete en la sesión.
- Adjuntar documentos adicionales para cada integrante del grupo.

Conceptualmente, cada proceso será una ventana diferente de la aplicación implementada utilizando la tecnología web de Openbravo Mobile, que, como se ha visto en el apartado 3.4.?, se implementa gracias al framework de javascript EnyoJS.

Dicha implementación deberá soportar:

- Compatibilidad con dispositivos móviles
- Funcionamiento offline.

Se requiere que los procesos sean muy intuitivos para una verdadera experiencia Business to Customer que no requiera ningún tipo de formación por parte de los usuarios, salvo el conocimiento de los flujos de trabajo normales.

También se requiere un proceso final en el back-end para enriquecer los datos introducidos desde la aplicación móvil y formalizar el documento mandándolo al ordenador central para su procesamiento. Este proceso sólo podrá ser accedido por trabajadores de Talent Partners desde las oficinas, no puede ser utilizado desde dispositivos móviles.

A mitad del desarrollo Talent Partners cambió de idea respecto al nombre de su aplicación, cambiándolo de VitaRatio a nAble, lo que provocó diversos cambios en el diseño de la aplicación.

4.1.2. Reparto de Trabajo.

En un primer acercamiento el desarrollo de la aplicación iba a ser realizado completamente por Openbravo. Sin embargo se estimó que realizar todo el desarrollo por una única empresa llevaría más tiempo del requerido por Talent Partners para el desarrollo de la aplicación, por lo que hubo que separar el desarrollo de la aplicación en dos, el desarrollo del backend por un lado y el desarrollo del front-end (la parte móvil de la aplicación) por otro.

Openbravo adquirió el desarrollo de la parte móvil de la aplicación, mientras que el desarrollo del back-end, una personalización del ERP de Openbravo, fue adquirido por Fugo Consulting, un partner de Openbravo con sede en la India.

Ambos desarrollos se llevaron a cabo de manera coordinada, orientados por TDS.

Figura 4-1 Logotipos de TDS y Fugo Consulting

4.2. Metodología de Trabajo

Durante todo el proceso de desarrollo de la aplicación se ha seguido una metodología de trabajo ágil, con un equipo de trabajo que en todo momento ha estado entre 3 y 5 personas.

En todo momento ha existido un jefe de proyecto que ha sido al mismo tiempo el encargado de las comunicaciones con TDS.

Al comienzo del desarrollo se realizaron una serie de reuniones para establecer los pasos a seguir y encauzar el desarrollo de manera que pudiera ser completado a tiempo.

Entre los puntos más relevantes se destacaron los siguientes:

- Encontrar una manera elegante de gestionar datos tanto online como offline, pues era uno de los requisitos más importantes establecidos por Talent Partners.
- Realizar un diseño funcional lo más completo posible para tener el alcance de la aplicación definido y cerrado. A lo largo del desarrollo se ha demostrado sin embargo que no ha sido posible terminar de definir el alcance pues constantemente ha habido cambios de especificaciones en la aplicación. Se entregaron periódicamente varios diseños funcionales pero todos fueron rechazados o se solicitaron modificaciones.
- Especificación del interfaz de usuario. Parte del diseño del interfaz venía especificado por una guía de estilo entregada por el cliente. Sin embargo aún faltaba por determinar la apariencia de los diversos menús de la aplicación.

Además de las reuniones iniciales, cada poco tiempo se han realizado pequeñas reuniones para revisar el estado de los desarrollos y replantear asignaciones de tareas, con el objetivo de completar el proyecto en el tiempo establecido.

4.2.1. Herramientas de trabajo utilizadas.

Las herramientas más importantes a la hora de desarrollar han sido el entorno de trabajo Eclipse IDE, el gestor de versiones Mercurial y el gestor de incidencias Jira.

4.2.1.1. Eclipse IDE

Trabajar con un entorno de desarrollo integrado permite al desarrollador tener en todo momento el código completo de la aplicación disponible para consulta. Además incluye una serie de funcionalidades que permiten agilizar el proceso de codificación.

Para empezar incluye un analizador de código, que muestra en tiempo real si el código que estamos desarrollando contiene errores, ya sean de tipo sintáctico o por referencias de datos. En muchos casos es capaz incluso de proponer la solución al problema encontrado, por ejemplo incluyendo sentencias de importación para resolver problemas de referencias entre clases.

También incluye una función de autocompletado de métodos, por lo que se puede saber en cada momento los métodos de los que dispone una clase sin tener que acudir a su implementación.

Eclipse incluye la posibilidad de incluir un servidor web propio, por lo que los cambios realizados mientras se desarrolla pueden ser vistos en tiempo real funcionando en la aplicación.

Los entornos de desarrollo implementan de forma transparente las operaciones de compilación y despliegue de las aplicaciones, eliminando la necesidad de realizar este trabajo de manera manual.

4.2.1.2. Mercurial

Openbravo utiliza Mercurial como software de gestión de versiones para mantener en todo momento un control sobre el código que se ha desarrollado.

Para el desarrollo de esta aplicación se ha utilizado Bitbucker.org como plataforma para almacenar los repositorios de la aplicación, y Mercurial como aplicación para gestionar dichos repositorios.

Las ventajas principales de utilizar software de control de versiones distribuido, como es el caso de Mercurial, son las siguientes:

- En todo momento existe una copia estable del repositorio. Cada desarrollador tiene una copia local del repositorio, sobre la cual realiza los cambios necesarios.
- Los desarrolladores no están limitados por la conectividad al repositorio remoto. Pueden desarrollar sin problemas en un entorno aislado y más adelante, al recuperar la conectividad con el servidor remoto, integrar los cambios realizados.
- Es posible examinar todas las versiones del código, ya que cada cambio realizado al repositorio queda registrado.
- Es posible saber quién ha realizado cada cambio en cada fichero

- No se almacenan copias de los ficheros que componen el repositorio. Sólo se almacenan las diferencias de una versión a la siguiente.

4.2.1.3. Jira

Jira es un gestor de incidencias perteneciente al grupo de aplicaciones Atlassian.

En el desarrollo de este proyecto se ha utilizado Jira para permitir al cliente reportar problemas encontrados en la aplicación y también realizar peticiones de nuevas funcionalidades.

Se ha utilizado una única instancia de Jira, gestionada por TDS, para que Talent Partners pudiera registrar incidencias de las dos partes de la aplicación, back-end y front-end.

Se ha utilizado especialmente al final de cada etapa de desarrollo y tras desplegar la aplicación a un entorno de pruebas del cliente para mantener un seguimiento de los problemas encontrados.

4.2.1.4. Otras herramientas

A la hora de desarrollar el proyecto se han utilizado una serie de herramientas adicionales para el correcto funcionamiento del equipo de desarrollo:

- Google Drive se ha utilizado como espacio compartido para almacenar diseños, actas de reuniones, bocetos, y, lo más importante, hojas de cálculo con seguimientos del estado del proyecto y con listados de tareas a realizar (resolución de bugs o implementación de nuevas funcionalidades).
- Skype se ha utilizado como medio de comunicación para las ocasiones en las que encuentros presenciales no fueran posibles, por ejemplo cuando alguno de los desarrolladores tenía que trabajar desde fuera de la oficina.
- Herramientas de diseño y escritura, como por ejemplo Adobe Photoshop o Microsoft Word, necesarios para realizar bocetos y diseños de las ventanas de la aplicación así como para escribir los informes y actas de las reuniones y otra serie de documentos necesarios en el proyecto.

4.2.2. Otras cuestiones a tener en cuenta.

A la hora de desarrollar se han tenido en cuenta ciertas buenas prácticas que han ayudado a mejorar la calidad del software desarrollado en la aplicación.

Por ejemplo por cada desarrollo realizado existía un responsable que se encargaba de revisar el código antes de integrarlo en el repositorio general. Esta práctica se conoce como Code Review (Revisión de Código) y permite encontrar problemas que el autor original del código ha podido omitir por estar demasiado familiarizado con su código.

Otra buena práctica es utilizar un software analizador de código que, en lugar de detectar errores, detecta problemas de calidad del código. En el caso de código javascript existe la aplicación JSLint, que comprueba y valida el código desarrollado.

Uno de los principales defectos que detecta JSLint es la declaración mal ubicada o no declaración de variables, que se considera código válido pero empeora la calidad del software.

4.3. La implementación

En esta sección se explicará cómo se ha estructurado la aplicación (qué módulos la componen), cuales son los procesos más importantes, la importancia y necesidad de las validaciones en la aplicación, y por último el proceso de entrega del producto y la atención prestada durante el periodo de garantía.

4.3.1. Estructura de la Aplicación

La aplicación utiliza como base Openbravo 3.0 en su versión (conocida en terminología de Openbravo como Maintenance Pack o pack de mantenimiento) 3.0MP23.

La aplicación se desarrollará creando 4 módulos nuevos y modificando el núcleo de Openbravo Mobile:

- El módulo Talent Vita Radio Main: Es el módulo principal y contiene la definición de los datos maestros y los procesos utilizados por la aplicación. Este módulo depende del núcleo de Openbravo.
- El módulo Talent Vita Radio Template: Es la plantilla requerida por Openbravo para realizar cambios sobre elementos pertenecientes al Core de Openbravo.

- El módulo Talent Vita Radio Mobile: Contiene todo el código de la aplicación móvil. Este módulo depende de los módulos Main y Mobile Core
- El módulo Talent Vita Radio Custom: Contiene formularios y vistas personalizadas. Depende de los módulos Main y Mobile.
- Openbravo Mobile Core: Es el módulo núcleo de toda aplicación móvil construida sobre Openbravo.

Los módulos Talent Vita Radio Main, Talent Vita Radio Template y Talent Vita Radio Custom son responsabilidad de Fugo Consulting, pues los módulos afectan principalmente al funcionamiento del backend.

Talent Vita Radio Mobile y Openbravo Mobile Core son responsabilidad de Openbravo pues afectan directamente al funcionamiento de la aplicación móvil.

4.3.2. Procesos Importantes.

4.3.2.1. Flujo General de Trabajo.

Mediante este esquema se pretende representar el flujo de trabajo estándar de la creación de un contrato con nAble:

1. Un productor se conecta a la aplicación y crea un proyecto nuevo.
2. Al proyecto se le asigna un cliente y otra serie de datos de carácter no obligatorio.
3. Dentro del proyecto el productor crea un contrato.
4. Se le asigna al contrato un performer (intérprete). Todo contrato tendrá asociado obligatoriamente un intérprete.
5. Se completan una serie de datos sobre el intérprete así como sobre los anuncios grabados (commercials).
6. Se guarda el contrato.
7. Se sincroniza la aplicación con el ERP, enviando al backend todos los datos del proyecto recién creado. El proceso de sincronización es uno de los procesos más importantes de la aplicación.
8. A partir de este punto se pueden seguir realizando modificaciones en el contrato o proyecto, pero siempre será necesario sincronizar la aplicación para que la información en el ERP y la información en la aplicación móvil sea consistente.

4.3.2.2. El proceso de sincronización.

Como ya se ha comentado en el apartado anterior el proceso de sincronización es uno de los procesos más importantes de la aplicación, y consiste en mantener los datos consistentes entre la aplicación móvil y la aplicación backend o de escritorio.

En nAble desde el principio ha existido el problema de la sincronización, dado que existen dos aplicaciones, el ERP (aplicación backend, aplicación de escritorio) y la aplicación móvil, relativamente independientes entre si, que trabajan sobre el mismo conjunto de datos, es decir, comparten la base de datos. En lo que sigue de sección denominaremos a las aplicaciones mencionadas Mobile y ERP).

Se barajaron diferentes opciones de sincronización, hasta que se dio con una solución que cumpliera los requisitos del cliente.

Antes de nada hubo problemas de terminología. Al proceso de sincronización se le llamó en un principio "Submit for Processing", pues se entendía el proceso de sincronización como el proceso de enviar los datos desde Mobile al ERP para su posterior procesamiento. Este término resultó ser el utilizado por Talent Partners para representar el proceso que:

- Disparaba la impresión de documentos pdf
- Insertaba los pdf en el sistema de documentos Onbase y
- Cambiaba el estado de los contratos a *submitted*.

Por ello se decidió renombrar el proceso de sincronización a "Sync with Vita".

Según el flujo especificado por el cliente, los usuarios de Mobile deben trabajar siempre con la versión final y actualizada de los datos. Además se debe permitir a los usuarios en la oficina (trabajando desde el ERP) revisar el trabajo hecho por los usuarios de Mobile para encontrar problemas y corregirlos y también para completar la información introducida.

Desde Openbravo se realizó la siguiente propuesta para el proceso de sincronización.

4.3.2.2.1. Propuesta de Openbravo para el proceso de sincronización

Los contratos contarán con un campo que indique qué aplicación es dueña del mismo. Dicho campo podrá contener los valores Mobile o ERP.

Los contratos podrán encontrarse en tres estados distintos: DRAFT (borrador), COMPLETE (completado) y IN PROGRESS (en progreso).

Habrá tres flujos de trabajo o casos de usuario diferentes:

Caso de Usuario 1. Usuario Mobile crea un contrato y lo sincroniza con el ERP.

1. Usuario Mobile crea un nuevo contrato en estado DRAFT. En este momento el dueño del contrato es Mobile.

2. Usuario Mobile edita el contrato en su aplicación móvil cuantas veces le haga falta.

3. Cuando el usuario Mobile termina de trabajar en el contrato lo envía al ERP usando un botón para lanzar el proceso de sincronización.

4. En este momento el contrato se transfiere al ERP, lo que significa que:

- El estado del contrato pasa de DRAFT a COMPLETE
- El dueño del contrato pasa a ser ERP
- El contrato pasa a ser de sólo lectura en Mobile.

5. El usuario Mobile tendrá acceso de sólo lectura al contrato durante un tiempo determinado para evitar sobrecargar el almacenamiento WebSQL del dispositivo.

Caso de usuario 2: Usuario ERP pasa a ser dueño de un contrato.

1. Después de que un usuario Mobile haya pulsado en *sincronizar* el usuario ERP recibe los contratos sincronizados con toda su información, incluyendo el usuario Mobile que ha realizado el proceso. En estos momentos los contratos se encuentran en estado “COMPLETE”.

2. En estos momentos el usuario ERP tiene varias opciones:

2.1. Revisar el contrato, validarlo y continuar el flujo de trabajo normal dentro del ERP.

2.2. Revisar el contrato y encontrar alguna inconsistencia que necesita ser revisada por el usuario Mobile. En este momento el usuario ERP puede cambiar el estado del contrato a “IN PROGRESS” y devolvérselo al usuario Mobile.

2.3. Revisar el contrato y encontrar alguna inconsistencia que tiene que revisar un usuario Mobile, diferente del usuario que creó el contrato. En este momento el usuario ERP puede cambiar la “propiedad” del contrato al nuevo usuario Mobile y cambiar el estado del contrato a “IN PROGRESS”

Caso de usuario 3: Usuario Mobile recibe la propiedad de un contrato IN PROGRESS

1. El usuario Mobile verá que algunos de sus contratos están en estado IN PROGRESS y necesitan ser revisados. Desde el ERP se habrá podido rellenar un campo de texto indicando qué necesita ser revisado.

2. Tras realizar las modificaciones necesarias el usuario Mobile volverá a realizar el proceso de sincronización devolviendo la propiedad del contrato al ERP y cambiando el estado del contrato de IN PROGRESS a COMPLETE.

3. A partir de este momento se iría de vuelta al caso 2, hasta que el contrato no necesite más modificaciones.

Algunas consideraciones.

1. Cada usuario Mobile verá solamente aquellos contratos de los cuales es dueño (bien porque los haya creado y estén pendientes de sincronizar o bien porque desde el ERP se le ha transferido la propiedad de los mismos). Al crear un contrato se asigna como dueño del mismo al creador; esta propiedad puede cambiarse en cualquier momento desde el ERP.

2. Para no sobrecargar el almacenamiento Web local de los dispositivos móviles, cada usuario podrá ver los contratos modificados en los últimos X días.

3. Existirá un mecanismo automatizado para liberar contratos IN PROGRESS que no lleguen a ser completados en un cierto periodo de tiempo. De este modo se asegura que todos los contratos que han sido sincronizados en algún momento acaben en estado COMPLETE.

La propuesta de Openbravo se rechazó, alegando que los dispositivos móviles no debían tener información de sólo lectura en ningún momento. Toda la información capturada durante una jornada de trabajo se considera mejor que la información almacenada en el ERP por lo que debería aplastarla para poder sincronizarse.

A continuación, TDS propuso una solución final para el proceso de sincronización, más simple que la propuesta inicial de Openbravo.

4.3.2.2.2. Propuesta de TDS para el proceso de sincronización

Los usuarios de nAble son, por norma genera, productores que trabajan par una agencia de publicidad. Por cada proyecto hay una sola persona encargada de gestionar los contratos y las sesiones de grabación incluidas en el mismo. El flujo de trabajo típico sería el siguiente:

1. Productor se conecta al ERP, crea un proyecto y en el mismo crea un contrato para cargar información por defecto de los intérpretes desde el

ordenador central. De este modo agiliza el proceso de creación de un contrato al no tener que introducir todos los datos a mano.

2. Antes de dejar la oficina, Productor se conecta a Mobile para sincronizar su dispositivo móvil con el ERP, cargando en la memoria local del dispositivo el proyecto recién creado con todos sus datos.

3. En el lugar de grabación, Productor utiliza Mobile para actualizar la información del intérprete en el contrato, y para crear un Session Commercial (con el cual se representa el anuncio grabado en esa sesión).

4. De vuelta en la oficina, Productor sincroniza el contenido del dispositivo móvil con el ERP, volcando los datos capturados durante el día al ERP.

Simplificación del proceso

Las sesiones de grabación de radio se realizan en un único día, y se gestionan por un único usuario (el cual está o bien online o bien offline). Por eso, se puede simplificar el flujo de trabajo anterior en el siguiente esquema:

1. Cada vez que el usuario se conecta a la aplicación Mobile de manera online (recordemos que existe la posibilidad de trabajar offline desde Mobile):

a. Determinar si existe en el dispositivo móvil (tablet, smartphone) nueva información, ya sea añadida o modificada, respecto a la información almacenada en el ERP. De ser así, automáticamente sincronizar con Vita Radio y mostrar un mensaje indicando que se ha realizado una sincronización y se ha añadido/modificado información.

b. Limpiar el almacenamiento local WebSQL

c. Descargar al dispositivo móvil toda la información actualizada desde la base de datos de Vita, y mostrar el mensaje correspondiente.

2. Adicionalmente, desde la Mobile y en cualquier momento, con la única condición de que la aplicación esté online, existirá un botón para lanzar el proceso “Sync with Vita” y ejecutar el proceso del paso 1. Este botón debería encontrarse en la ventana inicial de la aplicación.

Esta simplificación del proceso pone al productor en completo control de la información, y le proporciona la confianza de que sincronizar significa realmente sincronizar, es decir, cada vez que se lance ese proceso tanto el dispositivo como Vita tendrán la última versión de la información, siempre asumiendo que a la hora de sincronizar la información Mobile es mejor. Esto es así porque el productor recibe información directamente del intérprete y del equipo de grabación.

Como se ha descrito previamente, cuando un usuario realiza el proceso Submit for Processing, actualiza el estado del contrato a Submitted. A partir

de este punto el contrato pasa a ser de sólo lectura en Mobile. Nótese que al intentar sincronizar un contrato en estado Submitted los datos en la aplicación Mobile serán ignorados.

4.3.2.3. Creación de Plantillas y Creación de Re-records

Para agilizar la creación de contratos la aplicación ofrece dos procesos que reutilizan la información de otros contratos.

El proceso de plantillas permite guardar contratos marcados como plantilla, de forma que conservan gran parte de los datos del contrato original (incluyendo la información sobre el intérprete). A la hora de crear un contrato nuevo se ofrece la oportunidad de cargar una plantilla, de manera que no será necesario introducir todos los datos desde el principio.

Las plantillas se identifican por su nombre, de modo que no puede haber dos plantillas con el mismo nombre. Si se intenta guardar una plantilla con un nombre duplicado se mostrará un mensaje de confirmación para sobrescribir la plantilla existente.

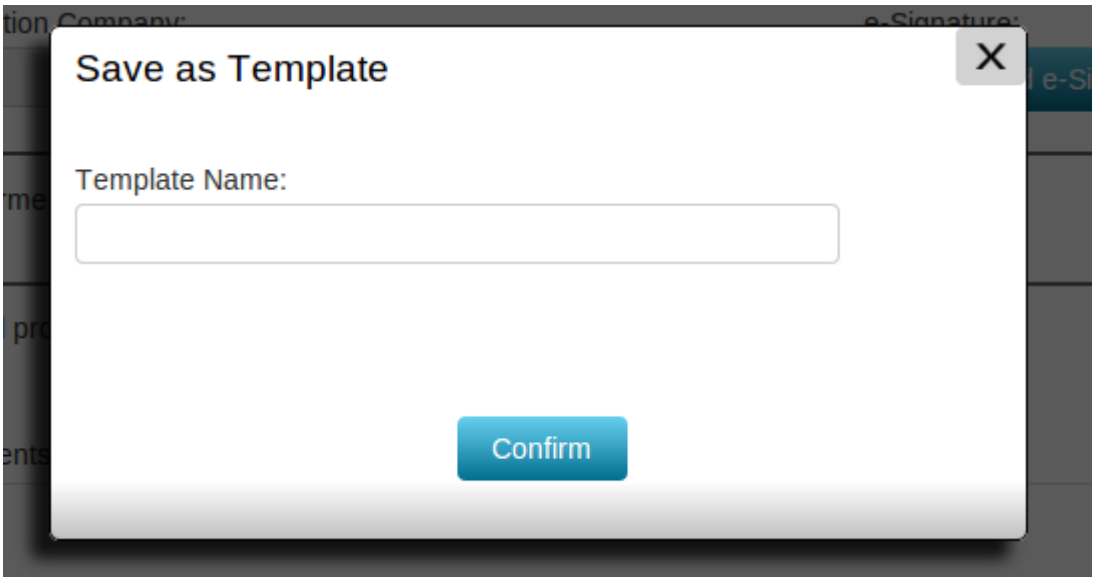

Figura 4-2 Pop-up de guardado de Plantillas

El proceso de creación de re-records es ligeramente diferente al proceso de creación de plantillas. Este proceso duplica un contrato y lo marca para representar que es un re-record. Este tipo de documento representa una repetición de una grabación, por ejemplo porque algo ha ido mal en la grabación original. A diferencia del proceso de creación de plantillas, al

crear un re-record se conservan todos los datos del contrato original, aunque pueden ser editados posteriormente.

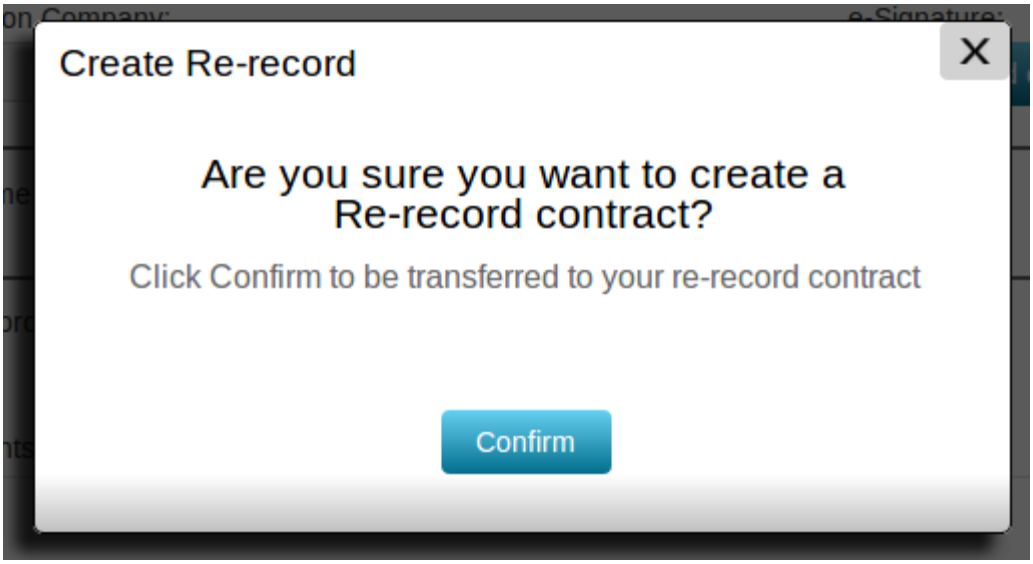

Figura 4-3 Pop-up de guardado de Re-records

4.3.3 Validaciones de Datos

La aplicación nAble es esencialmente una aplicación de captura de datos, un formulario con muchos campos, los cuales tienen que cumplir con diferentes validaciones y formatos.

Todos los datos capturados por la aplicación acaban en el ordenador central de Talent Partners, por lo que tienen que cumplir ciertas reglas para que sean compatibles con el formato utilizado en el ordenador central.

Las validaciones varían tanto en el tipo de validación como en el tipo de dato al que afectan. Ejemplos de las validaciones realizadas incluyen permitir o prohibir ciertos caracteres, obligar a los campos a tener un formato concreto, limitar la longitud o el valor máximo, o incluso modificar ciertos selectores según valores en otros campos.

A continuación se muestra una lista completa de las validaciones que se han incluido en la aplicación a nivel de campo:

- Una validación para aceptar únicamente caracteres de teléfono, es decir, números, paréntesis de apertura y cierre, y guiones simples.
- Una validación para caracteres alfanuméricos (letras y números)

- Una validación para caracteres alfabéticos (sólo letras)
- Una validación para aceptar cualquier carácter excepto el símbolo porcentaje o una comilla simple. Esta validación da a entender que los campos que la contengan formarán parte de alguna sentencia SQL. Si se permitieran comillas simples en una sentencia SQL la aplicación sería vulnerable a inyecciones SQL que pondrían en peligro la integridad y la seguridad del sistema.
- Una validación para permitir caracteres alfanuméricos, asteriscos y guiones simples.
- Una validación para prohibir insertar espacios en blanco (incluye tabuladores y saltos de línea).
- Una validación para permitir únicamente valores numéricos
- Una validación para exigir un formato tipo pin de cuatro dígitos.
- Una validación para exigir un formato tipo número de la seguridad social, con 9 caracteres que pueden ser dígitos o asteriscos (si el número está cifrado).
- Una validación para exigir un formato numérico, es decir, un texto formado por dígitos y como mucho un solo punto decimal.
- Una validación para no permitir más de dos decimales.
- Una validación para no permitir introducir espacios en blanco al inicio de un campo
- Una validación para permitir un solo espacio en blanco en todo el campo
- Una validación para exigir un formato de número de teléfono en formato americano, es decir, xxx-xxx-xxxx o (xxx)xxxxxxx
- Una validación para exigir un formato de correo electrónico.
- Una validación para fijar la longitud máxima del campo
- Una validación para fijar el valor máximo del campo.

Además de las validaciones a nivel de campo, la aplicación ha requerido diversas validaciones que incluyen:

- No duplicidad de valores a nivel de agencia o contrato, por ejemplo los identificadores.
- Limitar el valor de una fecha dentro de un rango.
- Convertir un campo en dato obligatorio si se ha introducido algún valor en él.
- Exigir que la suma de ciertos campos sea siempre igual a un valor.

Al principio cada validación se codificó directamente en la definición de cada tipo de datos, de modo que por ejemplo el tipo de campo *texto* tenía codificadas todas las validaciones de formatos, las validaciones de espacios en blanco, las validaciones de longitud, etc.

En cierto momento se vio que se estaba convirtiendo en un proceso muy complicado mantener las validaciones de ese modo. Por ejemplo, a la hora de crear un componente que heredara de otro con validaciones, y necesitara implementar su propia validación además de las originales, era necesario copiar todo el código otra vez.

Por eso, se creó un motor de validaciones que consiguió estandarizar las validaciones y minimizó la cantidad de código replicado.

Con este nuevo motor de validaciones, las validaciones pasaron a ser objetos con tres o cuatro atributos: el tipo de validación (a nivel global o de campo completo, a nivel de carácter, validación de longitud máxima y validación de valor máximo), la expresión regular que definía la validación (en caso de existir), el mensaje a mostrar en caso de incumplirse la validación y un valor no siempre presente que representaba el parámetro a pasarle al mensaje.

Por ejemplo, la validación de caracteres alfanuméricos pasó a tener el siguiente aspecto:

```
alphanumeric: {
    type: 'char',
    regExp: /[^0-9a-z\s]/i,
    message: 'TAVRMO_AlphanumericCharsOnly'
}
```

4.3.4 Doble entrega final en Diciembre y Marzo y Periodo de Garantía.

La entrega final del producto estaba programada para la primera semana de Diciembre de 2013. Tras dos semanas de intenso trabajo resolviendo los defectos encontrados en la aplicación, el código se congeló el 4 de diciembre y la aplicación fue entregada al cliente.

Sin embargo, el cliente no había probado la aplicación durante el periodo de pruebas establecido antes de la entrega final. Tras la entrega a primeros de diciembre, el cliente pidió una extensión del proyecto para poder probar la aplicación y resolver los defectos encontrados.

De este modo se abrió un nuevo periodo de desarrollo de tres meses, desde comienzos de año hasta finales de marzo, momento en el cual se realizó el despliegue de la segunda entrega final.

A partir de ese momento el cliente dispuso de un periodo de garantía de un mes, en el cual todos los defectos encontrados tenían que ser resueltos como si se trataran de errores de máxima prioridad.

REFERENCIAS

[1] (DiNucci, Darcy. 1999. Fragmented Future)

Todos los logotipos son propiedad de las empresas o productos respectivos que representan.

Las figuras 3-5 y 3-7 han sido extraídas de la Wiki de Openbravo (http://wiki.openbravo.com/wiki/Main_Page)

La figura 3-6 proviene de una instancia de Openbravo perteneciente al conjunto de instancias de prueba que se encuentran en http://livebuilds.openbravo.com/, concretamente de la instancia *Retail pgsql backend (https://livebuilds.openbravo.com/retail_pi_pgsql).*