Manual de uso del API suministrado por el Sistema de Notificaciones a las Entidades Emisoras

Octubre 2005

Versión 2.0.2

REVISIONES

Versión	Fecha	Objeto
2.0.2	05/10/05	• Se añade información acerca de los requisitos previos necesarios para el correcto funcionamiento del API de Notific@ (3.2)
2.0.1	20/07/05	• Se añade información acerca: o Del modelo Entidad-Relacion que se debe utilizar en la integración de una aplicación con Notific@ (4.1). o De un nuevo método constructor del API que permite indicar un objeto *properties* en vez de indicar la ruta de fichero de configuración (ver *Javadoc*).
2.0	24/05/05	• Se añade información acerca de: o El envío de notificaciones cifradas con el certificado X509 del destinatario. (3.5.2) o La obtención del certificado X509 de un conjunto de usuarios previamente autenticados contra el sistema. (3.6.4) o **El cambio en el archivo de configuración del API (mcsn.properties).** Se añade un nuevo método de obtención de la contraseña que protege a la clave privada del certificado de componte que firma las solicitudes. Anteriormente se indicaba en el fichero de configuración directamente la contraseña, ahora se permite además indicar, si se desea, una clase a partir de la cual se obtiene de una forma segura dicha contraseña. (3.2) o Los pasos a realizar para la conexión vía Https contra el servicio Web de Notific@. (3.2)
1.4	28/01/05	• Se añade información acerca de una nueva funcionalidad: o Solicitud que permite comprobar si un conjunto de usuarios están suscritos a un servicio. (3.6.3) • Se describe como se debe llevar a cabo una integración de una aplicación en Notific@ (4)
1.3	12/09/04	Se añade información acerca del fichero de configuración y se corrigen algunos errores en el documento.
1.2	02/03/04	Se añaden ejemplos para facilitar la integración del API en las aplicaciones de las Entidades Emisoras
1.1	20/11/03	Se añade información acerca del funcionamiento del Sistema de Notificaciones.

INDICE

1. Definición del Sistema de Notificaciones

Es un sistema para realizar el envío y seguimiento de **notificaciones telemáticas fehacientes**, con generación de evidencias comprobables de la entrega por el emisor y la recepción por el destinatario.

Conforme al Real Decreto 209/2003 de 21 de Febrero por el que se regulan los registros y las notificaciones telemáticas:

- Nuevo instrumento de comunicación ciudadano-Administración
- Consentimiento expreso del interesado
- Solo para los procedimientos expresamente señalados por el interesado
- Acreditación de fechas y horas de recepción del aviso de notificación y el acceso del interesado al contenido.

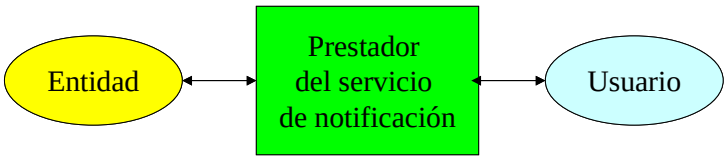
2. Arquitectura general del sistema

El sistema de notificación pondrá en relación a tres tipos de entidades que se representan en el esquema siguiente y que designaremos en lo sucesivo por:

Entidades: serán los Organismos de las Administraciones Públicas, las Empresas u otro tipo de entidades con personalidad jurídica propia que constituyen los clientes del servicio de notificaciones y que son los emisores de las notificaciones.

Usuarios: serán los ciudadanos, clientes, proveedores y en general los particulares destinatarios de las notificaciones.

Prestador del servicio de notificación: La Consejería de Justicia y Administración Pública de la Junta de Andalucía.



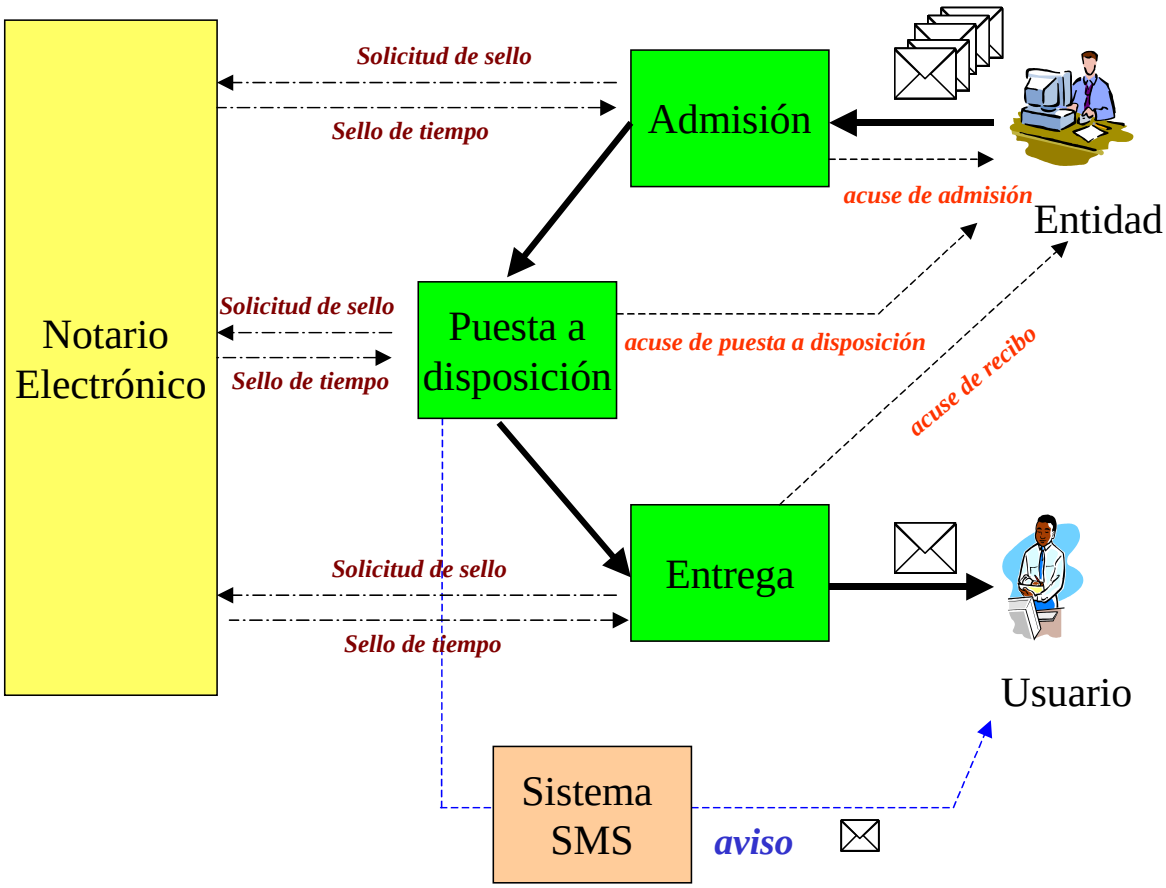
Los tratamientos asociados al envío y entrega de notificaciones pueden representarse mediante una serie de bloques o tratamientos que serán descritos posteriormente de forma mas detallada. De forma resumida son:

Proceso de admisión: Consiste en la entrada al sistema de lotes o remesas de notificaciones provenientes de una entidad determinada, las verificaciones y generación de mensajes de confirmación (acuse de envío)

Proceso de puesta a disposición: Tratamiento que procesa la remesa y deposita cada una de las notificaciones incluidas en ella en la dirección electrónica única del usuario destinatario. Genera acuses de “puesta a disposición” y, opcionalmente, puede generar avisos para los usuarios enviando un mensaje a un buzón de correo electrónico o a un teléfono móvil.

Proceso de entrega: El usuario podrá acceder a una notificación en particular, mediante una interfaz web apropiada. En el proceso se generarán "acuses de recibo" de las notificaciones accedidas.

En el siguiente gráfico se muestra un esquema del proceso seguido desde que una entidad emisora envía una remesa (conjunto de notificaciones) hasta que un usuario destinatario recibe una notificación contenida en la remesa. Como se puede observar como cada uno de los procesos se comunica con el Notario Electrónico para la generación de un sello de tiempo, de tal modo que se tenga constancia de que un proceso finalizó correctamente en un momento dado en el tiempo.

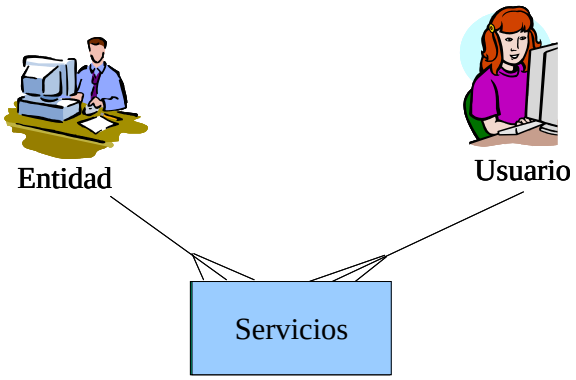


2.1 Servicios o suscripciones

Una funcionalidad importante del sistema, es el concepto de "servicio". Un servicio puede ser definido como el nexo de unión entre la entidad emisora y el Usuario. A través de este, el usuario es capaz de recibir notificaciones procedentes de la Entidad Emisora. Como es lógico antes de que un usuario pueda recibir estas notificaciones es necesario que este se suscriba al servicio asociado a la Entidad Emisora por propia iniciativa a través del interfaz web del usuario o suscribiéndole la Entidad Emisora.

El usuario se suscribirá a determinados servicios de entre la lista de los servicios disponibles que le presentará el sistema, y las entidades emisoras a su vez podrán enviar notificaciones correspondientes a aquellos servicios autorizados por el

administrador del sistema. En definitiva: una entidad podrá notificar bajo diferentes servicios (uno único en muchos casos), y los usuarios podrán suscribirse voluntariamente a un número indeterminado de servicios de notificación.



3. Funcionalidad del API de Entidades Emisoras

Steria ha desarrollado un API programado en JAVA que permite interactuar con el Sistema de Notificaciones a través del protocolo SOAP (*Simple Object Access Protocol*) contra un *WebService*. El API está compuesto por una serie de métodos que permiten realizar las siguientes funcionalidades:

- Suscripción de un Usuario a un servicio asociado a la Entidad Emisora.
- Baja de un Usuario de un servicio asociado a la Entidad Emisora.
- Construcción de una remesa de notificaciones cifradas o sin cifrar y envío de la misma a un servicio asociado, previamente contratado con el prestador de servicios de notificación.
- Envío de una circular a todos los Abonados suscritos a un servicio. Esta circular no podrá estar cifrada con la clave pública de los destinatarios.
- Obtención de información acerca de:
    - Los usuarios que se han dado de alta por iniciativa propia a un servicio asociado de la Entidad Emisora, entre una fecha inicial y final.
    - Los usuarios que se han dado de baja por iniciativa propia de un servicio asociado de la Entidad Emisora, entre una fecha inicial y final.
    - Las remesas enviadas al sistema de notificaciones. Se podrá obtener información sobre la admisión de la remesa, la puesta a disposición de las notificaciones contenidas en la remesa, la lectura o no de una notificación, errores, etc.

Todas las solicitudes que la Entidad Emisora envía al Sistema de Notificaciones a través del API estarán firmadas por un certificado de componente que la Entidad Emisora deberá solicitar a la FNMT y que deberá ser indicado en el archivo de configuración del API. Este archivo se explicará en el siguiente punto.

3.1 Descripción del contenido del CD entregrado con todo lo necesario para utilizar el cliente de Sistema de Notificaciones

La estructura de directorios es la siguiente:

1. Directorio **lib**, donde se encuentran todas las librerías o ficheros jar necesarios para poder ejecutar la aplicación. El API de Entidades Emisoras se encuentra en la librería *ClientSN.jar*.
2. Directorio **configuración** con los ficheros de configuración del Cliente de Sistema de Notificaciones. Estos ficheros son: a) *mcsn.properties*, para los parámetros del cliente y b) *mcnelog4jConfig.xml*, para la configuración del log. Pueden renombrarse y reconfigurarse.
3. Directorio **doc**, donde se encuentran los ficheros html con la documentación del API desarrollado, y este documento. Los documentos html se encuentran en un directorio llamado html, donde se encuentra un archivo index.html que da entrada a la ayuda.
4. Directorio **Ejemplos**, donde aparecen un grupo de programas de ejemplo de uso del API denominados *EnviarRemesa.java, EnviarNotificacionCifrada.java y CSNTest.java*

3.2 Requisitos previos

A continuación se indican los requisitos previos que son necesarios tener en cuenta y realizar para el correcto funcionamiento del API.

1. Incorporar en el *classpath* del sistema o servidor de aplicaciones todas las librerías que se encuentran en el directorio ***lib***.
2. Debido a que el API de Notific@ firma con un certificado de componente, que tiene una clave privada de más de 1024 bits, las solicitudes *SOAP* que se envían al Servidor y los posibles *PDFs* que se quieren cifrar y firmar, es necesario que se meta un parche en el *JDK* instalado en el sistema para eliminar la restricción de firma con certificados de mas de 512 bits. Para más información y descarga acceda a la siguiente URL:
   - **Para el J2SE1.4**:

     http://java.sun.com/j2se/1.4.2/download.html

     Sección: Unlimited Strength Jurisdiction Policy Files 1.4.2
   - **Para el J2SE1.5**:

     http://java.sun.com/j2se/1.5.0/download.jsp

     Sección: Unlimited Strength Jurisdiction Policy Files 1.5.0
3. En el caso de configurar el API para una conexión *https* contra el servidor será necesario importar el certificado (clave pública del servidor Web de Notific@) en el *keystore* de por defecto de la máquina virtual java denominado *cacerts*, y que está ubicado en $JAVA_NOME/jre/lib/security.

   El comando a ejecutar para importar el certificado es:

Keytool –import –keystore cacerts –storepass changeit –alias Notifica –file notifica.cer

3.3 Instanciación del API

El programador que desee invocar uno de los métodos funcionales del API deberá previamente instanciar la clase *MCSN*, ubicada en el paquete *notificaciones.cliente.api*.

Un ejemplo de instanciación sería el siguiente:

Notificaciones.cliente.api.MCSN mcsn=notificaciones.cliente.api.MCSN("C:/mcsn.properties, true)

Como se puede observar en la anterior línea la construcción de un objeto de tipo *MCSN*, conlleva el paso de los parámetros siguientes:

- La ruta absoluta del archivo de configuración del módulo cliente del sistema de notificaciones.
- Parámetro lógico. Indica si el API debe configurar el logger LOG4J o por el contrario la aplicación que invoca el API ya ha configurado el mismo el logger.

Existe otro constructor del API (clase MCSN) que permite, en vez de indicar la ruta del fichero de configuración indicar una instancia de un objeto *Properties* previamente cargado con los parámetros del fichero de configuración. Este constructor es útil cuando se va a utilizar el API muchas veces y no se desea leer el fichero de configuración cada vez.

Archivo de configuración del módulo cliente del sistema de notificaciones

En el siguiente cuadro se muestra un ejemplo de archivo de configuración.

```
#Características de la conexión con el sistema de notificaciones.
protocolo=http
direccion_ip=194.224.161.244
puerto=80
path_acceso=axis/*/services/ServicioWEBSN


# ------------------
# Conexión con proxy
# ------------------
# Con conexión proxy a true indica que el acceso es via proxy.
conexionproxy = false
# Nombre de servidor proxy o dirección IP:
proxyhost = nombre del HOST proxy o dirección IP
# Puerto del servidor proxy:
proxyport = 8080
# login conexión al proxy (vacio=>sin autenticación):
proxylogin =
```

 # password conexión con proxy:

proxypassword =

 # ----------

 # Configuración del Log de trazas

 # ----------

 # Fichero de configuración log para la salida con el logger de Log4j

xml_log=C:/ mcneLog4jConfig.xml

 # ----------

 # PKCS#12 para la firma de las solicitudes SOAP

 # ----------

 # Fichero PKCS#12 que contiene la pareja de claves

pkcs12.archivo = C:/SoapCertificate.p12

 # Metodo de obtención de la contraseña (clase o propiedad). La contraseña se podrá obtener

 # a través del atributo pkcs12.pass o desde el método getPrivateKey() de una clase que

 # implementa el interfaz IKeySec (Ver descripcion más abajo).

pkcs12.pass.metodo=clase

 # Contraseña del PKCS#12 que protege la clave privada.(Metodo propiedad)

pkcs12.pass = xmlsignature123

 # Nombre de la clase junto con el paquete que cumple el interfaz IKeySec (Ver Javadoc) y que permite implementar el método getPrivateKey() encargado de obtener de una fuente segura la contraseña. (Método clase)

 # Esta clase deberá encontrarse en el classpath del sistema.

pkcs12.clasepwd = mi.paquete.seguridad.PassPrivateKey

Parámetros importantes de la configuración del MCSN

Nombre	Descripción	Posibles valores
protocolo	Indica el protocolo mediante el cual el módulo se comunicará con el Sistema	• http • https (**Ver Nota mas abajo**)
dirección_ip	Dirección IP donde se encuentra el servidor del sistema de notificaciones	• Dirección IP del servidor
puerto	Indica el puerto de escucha del servidor web del sistema de notificaciones que acepta las solicitudes SOAP. Irá en función del	• puerto no seguro de http

	protocolo de comunicación utilizado.	• puerto seguro de https (**Ver Nota mas abajo**)
path_acceso	Path de acceso al Servicio WEB que atiende las peticiones del Módulo cliente del Sistema de Notificaciones	• Dejar el que viene por defecto. **NO TOCAR**
pkcs12.archivo	Todas las solicitudes enviadas al sistema van firmadas. Por ello es necesario que se indique la ruta absoluta del fichero PKCS#12 que contiene la pareja de claves.	
pkcs12.pass.metodo	Metodo de obtención de la contraseña. Esta se podrá obtener a través del atributo pkcs12.pass o desde el método getPrivateKey() de una clase que implemente el interfaz IKeySec. Ver propiedad pkcs12.clasepwd	• clase • propiedad
pkcs12.pass	Contraseña de acceso a la clave privada del PKCS#12	
pkcs12.clasepwd	Clase que cumple el interfaz IKeySec y que permite implementar el método getPrivateKey(), encargado de obtener de una fuente segura la contraseña. Para más información ver en el **Javadoc** la clase (**notificaciones.cliente.security.IKeySec**)	

Nota

En el caso de configurar el API para una conexión *https* contra el servidor será necesario importar el certificado (clave pública) en el *keystore* de por defecto de la máquina virtual java denominado *cacerts*, y que está ubicado en $JAVA_NOME/jre/lib/security.

El comando a ejecutar para importar el certificado es:

Keytool –import –keystore cacerts –storepass changeit –alias Notifica –file notifica.cer

El parámetro de configuración *xml_log* permite indicar la ruta absoluta de un fichero en formato XML que permite configurar el logger log4j utilizado en el módulo cliente. Un ejemplo de este fichero podría ser el siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
```

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

 <appender name="appender" class="org.apache.log4j.FileAppender">
  <param name="File" value="C:\\logs\\mcneLog4j.txt"/>
  <param name="Append" value="false"/>
  <layout class="org.apache.log4j.PatternLayout">
   <param name="ConversionPattern" value="%d [%t] %p - %m%n"/>
  </layout>
 </appender>

 <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">
        <layout class="org.apache.log4j.PatternLayout">
                <param name="ConversionPattern" value="%d{HH:mm:ss} [%c] %p - %m%n"/>
        </layout>
</appender>

 <root>
  <priority value ="info"/>
  <appender-ref ref="STDOUT"/>
 </root>

</log4j:configuration>
```

La forma en la que la aplicación mostrará información puede modificarse cambiando los contenidos del fichero. Como un breve resumen se puede considerar lo siguiente:

- La etiqueta "**appender**" indicará cómo y dónde se muestra la salida textual.

  En el ejemplo anterior, existen dos etiquetas "appender", una para enviar la salida hacia un fichero cuyo path también se especifica, y otro que utiliza la salida estándar, o por pantalla.
- La etiqueta ConversionPattern especifica el formato utilizado por el logger para mostrar la información.
- Por último, con la etiqueta "root" se configuran dos cosas: el nivel del logger (en este caso "info"), y el "appender" de los definidos que se selecciona, en este caso STDOUT, o salida estándar. Dentro del código de la aplicación habrá escritas diferentes llamadas al logger, cada una invocando una de sus funciones que utilizará alguno de los niveles con los que el logger muestra la información: DEBUG, INFO, WARN, ERROR y FATAL (ALL y OFF). El comportamiento del logger es jerárquico conforme al nivel con el que se haya configurado, de tal

modo que mostrará todos los mensajes de ese nivel y de los niveles que estén por debajo. Por tanto, con el nivel info, saldrán también los mensajes error, warn y fatal.

Para obtener información más detallada del comportamiento de los logger, consultar la dirección http://jakarta.apache.org/log4j/docs/documentacion.html

3.4 Suscripción de un usuario a un servicio.

El proceso de suscripción de un Abonado a un servicio puede ser realizado de dos formas diferentes:

1. Por iniciativa del usuario desde el interfaz web suministrado por el sistema de notificaciones.
2. Desde una aplicación de la consejería utilizando para ello el API de Entidades Emisoras suministrado por Steria.

En el caso de la segunda opción es necesario tener en cuenta los siguientes puntos:

- Si el usuario no está dado de alta en el sistema en el momento de suscribirlo a un servicio, el sistema lo dará de alta automáticamente.
- Es importante antes de poder dar de alta a usuarios en servicios disponer de los códigos de servicios proporcionados por el Administrador del Sistema a la Entidad Emisora.
- Los datos que debe disponer la entidad acerca del Usuario para realizar el alta en un servicio asociado son:
  - Nombre
  - Apellidos
  - Identificador de Abonado. A través de este, el sistema identificará al usuario. Este identificador es devuelto por el servidor de autenticación de @firma en el proceso de autenticación de un usuario vía servicio Web o método GET.
  - Teléfono móvil. OPCIONAL. En el caso de que se indique se le enviará un mensaje SMS a este móvil, indicando la puesta a disposición de una notificación por parte del sistema de notificaciones.
  - Correo Electrónico. OPCIONAL. En el caso de que se indique se le enviará un correo electrónico a esta dirección, indicando la puesta a disposición de una notificación por parte del sistema de notificaciones.
  - El FormReference y el TransactionID devuelto por el Servidor de Firma, en el momento de firmar el usuario la aceptación de recepción de determinadas notificaciones por vía telemática, según lo dispuesto en el Real Decreto 209/2003 de 21 de febrero. En el caso de que la aplicación de la Entidad Emisora tenga su propio sistema de firma, deberá suministrar el array de bytes de la firma generada por el usuario aceptando los términos de suscripción a un servicio o procedimiento.

El teléfono y el correo electrónico podrán ser posteriormente modificados por el Abonado desde la interfaz web que suministra el sistema para el acceso a las notificaciones recibidas.

NOTA

Es responsabilidad de la Entidad Emisora llevar un control de todos los abonados que ha dado de alta en un servicio, almacenando el identificador de abonado generado y el código de servicio en el que está dado de alta.

Un ejemplo de código JAVA que permite el alta de un abonado en el sistema y su suscripción a un servicio sería el siguiente:

```
Instanciamos el API de Entidades Emisoras
  notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);
Creamos un Abonado destinatario de la notificación
  Abonado abonado = new Abonado("09678456UPAZ ROBR");
  abonado.setNombre("Raúl");
  abonado.setApellidos("de la Paz Robledo");
  abonado.setTelefonoMovil("654678909");
  abonado.setEmail("luis @yahoo.es");
  FirmaInf firma = new FirmaInf("formreference","TransactionID");
Enviamos la solicitud de Alta de un usuario en el servicio con código 1
  mcsn.solicitarAltaAbonado(abonado, firma, 1);
```

3.5 Baja de un usuario de un servicio.

A través de esta funcionalidad la Entidad Emisora podrá dar de baja un usuario de un servicio suscrito previamente. Para ello la Entidad Emisora deberá suministrar al API un objeto de tipo Abonado donde se haya indicado el identificador del usuario (Ejem. *09678456UPAZ ROBR*).

Un ejemplo de código JAVA que permite la baja de un abonado de un servicio sería el siguiente:

```
Instanciamos el API de Entidades Emisoras
  notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);
Creamos un objeto Abonado (Usuario que queremos dar de baja)
 Abonado abonado = new Abonado("09678456UPAZ ROBR");
Enviamos la solicitud de Alta de un usuario en el servicio con código 1
  mcsn.solicitarBajaAbonado(abonado,1);
```

3.6 Envío de Remesas de notificaciones

Una entidad emisora podrá construir una remesa utilizando las clases suministradas en el API dado por el prestador de servicios de notificación. Una vez construida podrá enviarla al sistema de notificaciones utilizando el método de la clase MCSN:

enviarRemesa(Remesa remesa).

Las notificaciones contenidas en la remesa a enviar podrán cifrarse y firmarse en función de la necesidad que tenga la Entidad Emisora. Si la notificación a enviar es informativa y no requiere de máxima confidencialidad no será necesario que se cifre, pudiéndola enviar en claro. Sin embargo, para aquellas notificaciones que requieran de máxima confidencialidad por su contenido, se podrán cifrar con la clave pública del certificado del destinatario de la notificación.

Una notificación cifrada es para el Sistema de Notificaciones una notificación que contiene adjuntos de tipo PDF cifrados con la clave pública del certificado del destinatario y firmados por el certificado de componente asociado a la Entidad Emisora. El cuerpo, el asunto y otros adjuntos (no PDF) de la notificación van sin cifrar y por lo tanto en ellos no se debería meter información relevante o importante.

3.6.1 Envío de una remesa con notificaciones sin cifrar

A continuación se muestra un ejemplo de envío de una remesa que contiene una notificación sin cifrar.

```
Instanciamos el API de Entidades Emisoras
  notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);
Creamos un Abonado destinatario de la notificación
  Abonado abonado = new Abonado("09678456UPAZ ROBR");
Leemos el fichero que vamos a adjuntar a la notificación
  java.io.File adjunto = new java.io.File("C:/adjunto.pdf");
  if (!adjunto.exists())
       throw new Exception("No se ha encontrado el adjunto en " + pathAdjunto);
  java.io.FileInputStream fis = new java.io.FileInputStream(adjunto);
  java.io.ByteArrayOutputStream baos = new java.io.ByteArrayOutputStream();
  byte[] buffer = new byte[1024];
  int i = 0;
  while ((i = fis.read(buffer)) != -1){
        baos.write(buffer,0, i);
  }
  fis.close();
Construimos el adjunto
   Adjunto adj1 = new Adjunto(baos.toByteArray(),adjunto.getName());
   baos.close();
Construimos la notificación
   Notificacion notif1 = new Notificacion(false);
   notif1.setAsunto("Información");
  notif1.setCuerpo("Discurso del Presidente del CC.AA. de Andalucía ");
  notif1.addAdjunto(adj1) ; //Podemos adjuntar tantos adjuntos como queramos
```

```
notif1.addDestinatario(abonado); //Podemos añadir tantos destinatarios como queramos
Construimos la remesa que contiene el adjunto creado
Remesa remesa =new Remesa(1);
remesa.addNotificacion(notif1);
Enviamos la remesa al Sistema de Notificaciones
mcsn.enviarRemesa(remesa);
```

Notas:

- El texto del mensaje puede contener palabras con tildes y URLs completas. Además el sistema permite el uso de retornos de carro "\n", tabuladores "\t" y caracteres especiales como el signo de mayor ">" y menor "<".
- Si se desea enviar una notificación a todos los usuarios suscritos a un servicio, es decir una ***circular***, se podrá realizar esta acción tan sólo indicando *true* en el constructor de la notificación.

3.6.2 Envió de remesa con notificaciones cifradas

Como se ha comentado con anterioridad solo se pueden cifrar y firmar los adjuntos de tipo PDF[1] adjuntados en la notificación. Estos PDFs cifrados y firmados solo podrán ser leídos por el destinatario con el Acrobat Reader 6.0 o superior, ya que esta versión de Acrobat es la única que tiene los componentes necesarios para poder descifrar y verificar la firma que contiene.

Todos los adjuntos PDF que se cifren irán, además de cifrados, firmados electrónicamente por el certificado de componente asociado a la Entidad Emisora e indicado en el fichero de configuración del módulo cliente del sistema de notificaciones.

Para realizar la firma y el cifrado de PDFs, el API de Entidades Emisoras utiliza una librería de libre distribución denominado iText, el cual ha sido modificado para incorporarle la funcionalidad de cifrado. Además, con el objetivo de que la aplicación pueda utilizar iText y futuras versiones para generar PDFs o leerlos, se ha modificado el nombre del paquete de '*com.lowagie.text'* a '*st.com.lowagie.text'*, de tal manera que no haya ningún conflicto con la librería que viene con el API de Entidades Emisoras.

Descripción de iText

iText está formado por una serie de clases tremendamente útiles para quienes necesitan generar documentos en un formato portable, es decir totalmente independiente de la plataforma desde la que se consultan, sean textos, imágenes, listas o tablas, no importa de momento que tengas iText.

iText permite generar documentos PDF, XML, HTML o RTF al vuelo, controlando de forma precisa el aspecto del documento obtenido. Del mismo modo, podrás parsear documentos XML y convertirlos en cualquiera de los formatos mencionados arriba.

[1] La firma y el cifrado del PDF están embebidos dentro del mismo y cumple con la especificación 1.5 de Adobe.

Es ideal también para combinar múltiples documentos PDF en un nuevo fichero PDF pero, los que más van a valorar sus capacidades son los desarrolladores que tendrán la posibilidad de implementar nuevas capacidades a sus aplicaciones, gracias a la generación dinámica de documentos PDF.

Si se desea más información acerca de esta librería, puede consultarse en la dirección http://www.lowagie.com/iText/

Es importante que se le indique al destinatario en el texto del cuerpo, que la notificación va cifrada con su certificado digital y que es necesario que siempre conserve el certificado digital aunque haya caducado o revocado, ya que si el usuario elimina la clave privada asociada a la clave pública con la que se cifró el PDF, no podrá leerlo de nuevo.

Un ejemplo de código JAVA que permite la construcción y envío de una remesa con notificaciones cifradas podría ser el siguiente:

```
Instanciamos el API de Entidades Emisoras
notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);

Creamos el o los destinatarios de la notificación
CriptoAbonado abonado1 = new CriptoAbonado("09678456UPAZ ROBR");

Leemos el certificado del Abonado con el que se cifrará el adjunto PDF y lo asociamos al objeto Abonado
CertificateFactory fact = CertificateFactory.getInstance("X.509");
FileInputStream fisCer = new FileInputStream("configuracion/luis.cer");
X509Certificate cert = (X509Certificate)fact.generateCertificate(fisCer);
abonado1.setCertificadoDER(cert.getEncoded());

Leemos el PDF que vamos a adjuntar a la notificación y que cifraremos y firmaremos
java.io.File adjunto = new java.io.File("C:/infrelevan.pdf");
if (!adjunto.exists())
        throw new Exception("No se ha encontrado el adjunto en " + pathAdjunto1);
java.io.FileInputStream fis = new java.io.FileInputStream(adjunto);
java.io.ByteArrayOutputStream baos = new java.io.ByteArrayOutputStream();
byte[] buffer = new byte[1024];
int i = 0;
while ((i = fis.read(buffer)) != -1){
    baos.write(buffer,0, i);
}
```

```
fis.close();
baos.close();

Construimos el adjunto
Adjunto adj1 = new Adjunto(baos.toByteArray(),adjunto.getName());
Construimos la notificación cifrada
NotificacionCifrada notif1 = new NotificacionCifrada();
notif1.setAsunto("Información importante");
notif1.setCuerpo("Buenas");
Antes de adjuntar un PDF es necesario indicar los destinatarios
notif1.addDestinatario(abonado1);
Adjuntamos los PDFs creados con anterioridad a la notificación.
notif1.addPDFParaFirmarYCifrar(adj1); //Podemos adjuntar tantos adjuntos PDF como queramos
Firmamos y ciframos los Adjuntos PDF (La firma la podemos hacer visible en el documento PDF o no)
notif1.firmarYCifrarAdjuntosPDF("Autenticación","Sevilla",true,new Rectangle(100, 100, 200, 200),1);
Construimos la remesa que contiene el adjunto creado
Remesa remesa =new Remesa(1);
remesa.addNotificacion(notif1);
logger.info("Remesa construida.");
Finalmente enviamos la remesa
ES IMPORTANTE GUARDAR EL NÚMERO DE REMESA CON EL OBJETIVO DE POSTERIORMENTE /SACAR INFORMACIÓN DE SU PROCESAMIENTO.
logger.info("num remesa" + mcsn.enviarRemesa(remesa));
```

Notas:

- El texto del mensaje puede contener palabras con tildes y URLs completas. Además el sistema permite el uso de retornos de carro "\n", tabuladores "\t" y caracteres especiales como el signo de mayor ">" y menor "<".
- En una remesa puede haber tanto notificaciones cifradas como no cifradas.
- **Es importante que se le indique al destinatario en el texto del cuerpo, que la notificación va cifrada con su certificado digital y que es necesario que siempre conserve el certificado digital aunque haya caducado o revocado, ya que si el usuario elimina la clave privada asociada a la clave pública con la que se cifró el PDF, no podrá abrirlo.**

3.7 Sistema de información para entidades

Dentro de este subsistema se agrupan los procesos encaminados a extraer y servir informaciones del sistema de notificación a las entidades que lo hayan contratado.

Las informaciones disponibles para las entidades serán:

- Usuarios suscritos por iniciativa propia, entre una fecha inicial y final, a servicios de notificaciones contratados por la entidad.
- Usuarios dados de baja por iniciativa propia, entre una fecha inicial y final, a servicios de notificaciones contratados por la entidad.
- Acuses de admisión. Mediante estos acuses se informará tanto de las remesas admitidas por el sistema de notificación como de los posibles errores detectados en la admisión.
- Acuses de puesta a disposición de los mensajes. Asimismo se generarán errores de puesta a disposición de aquellos mensajes que no se puedan depositar en el buzón del usuario, indicando el error detectado (usuario desconocido, usuario no suscrito al servicio, etc).
- Acuses de recibo de los mensajes de aquellos servicios que lo requieran.
- Estado de un conjunto de usuarios en un servicio asociado.
- Certificado X509 de un conjunto de usuarios, obtenido por el sistema en el momento de autenticación del usuario en el Sistema de Notificaciones.

3.7.1 Información acerca de los usuarios dados de alta en un servicio

Existe en el Sistema de Notificaciones la posibilidad de que un Usuario previamente dado de alta en el sistema opte por suscribirse a un servicio específico asociado a una Entidad Emisora. Pues bien, esta entidad podrá conocer que nuevos abonados se han dado de alta en un servicio asociado invocando un método del API. El método en cuestión será el siguiente:

AbonadoInf[] solicitarInformacionAltasAbonado(Date fechaIni, Date fechaFin,int cod_Servicio)

En la invocación del anterior método será necesario que la Entidad Emisora indique el código de servicio, suministrado por el Administrador del Sistema de Notificaciones, del cual se quieren obtener los Abonados dados de alta por iniciativa propia. Además será necesario que indique el rango de fechas en los que los abonados se dieron de alta en el servicio indicado.

Como resultado de la invocación del método se obtendrá información de los usuarios dados alta por propia iniciativa. Dicha información estará compuesta por lo siguiente:

- Nombre del usuario
- Apellidos del usuario
- Identificador del usuario en el sistema

3.7.2 Información acerca de los usuarios dados de baja de un servicio

Existe en el Sistema de Notificaciones la posibilidad de que un Usuario previamente dado de alta en el sistema opte por darse de baja de un servicio específico asociado a una Entidad Emisora. Pues bien, esta entidad podrá conocer que abonados se han dado de baja de un servicio asociado invocando un método del API. El método en cuestión será el siguiente:

AbonadoInf[] solicitarInformacionBajasAbonado(Date fechaIni, Date fechaFin,int cod_Servicio)

En la invocación del anterior método será necesario que la Entidad Emisora indique el código de servicio, suministrado por el Administrador del Sistema de Notificaciones, del cual se quieren obtener los Abonados dados de alta por iniciativa propia. Además será necesario que indique el rango de fechas en los que los abonados se dieron de baja del servicio indicado.

Como resultado de la invocación del método se obtendrá información de los usuarios dados baja por iniciativa propia. Dicha información estará compuesta por la siguiente:

- Nombre del usuario
- Apellidos del usuario
- Teléfono móvil (Opcional). Puede ser que no desee que se le notifique por teléfono la puesta a disposición de una notificación.
- Correo electrónico (Opcional)

3.7.3 Información acerca del estado de un conjunto de abonados en un servicio

Las entidades emisoras podrán en todo momento conocer si un conjunto de usuarios están suscritos a un servicio asociado a la Entidad, gracias a la invocación del método siguiente:

public int[] solicitarEstadoAbonadoServicio(String[] anagramasAbons, int cod_Servicio)

En la invocación del anterior método será necesario que la Entidad Emisora indique los siguientes parámetros:

- AnagramasAbons: Es el conjunto de anagramas fiscales que son los identificadores unívocos de los usuarios que queremos conocer su estado en un servicio.
- Código de servicio, suministrado por el Administrador del Sistema de Notificaciones, del cual se quiere comprobar si el usuario está suscrito.

Como resultado de la invocación del método se obtendrá un *array* de tipo entero por cada anagrama consultado. Dependiendo de su valor indicará:

- **-1**, El usuario **no** está dado de alta en el sistema
- **0**, El usuario **no** está suscrito al servicio indicado.
- **1**, El usuario **está suscrito** al servicio indicado

Un ejemplo de código JAVA que permite la invocación de esta funcionalidad sería el siguiente:

```
Instanciamos el API de Entidades Emisoras
  notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);
Enviamos la solicitud de estado de un abonado en el servicio con código 1
 String[] ids = new String[1];
  Ids[0]= “09678456UPAZ ROBR”;
  Int[] res = mcsn.solicitarEstadoAbonadosServicio(ids,1);
```

```
Procesamos el resultado
  if (res[0] == -1)
       logger.info("El usuario NO está dado de alta en el sistema");
 else if (res[0] == 0)
       logger.info("El usuario NO está suscrito en el SERVICIO " + 1 + ", pero si esta dado de alta en el sistema");
  else if (res[0] == 1)
          logger.info("El usuario SI está suscrito al SERVICIO " + 1 );
```

3.7.4 Obtención del certificado X509 de un conjunto de usuarios

Para poder cifrar PDFs es necesario que la Entidad Emisora tenga el certificado X509 del destinatario y la forma de obtenerlo es responsabilidad suya, sin embargo el Sistema de Notificaciones tiene la cortesía de suministrar un servicio a través del API que permite obtener el certificado X509 de un conjunto de usuarios, siempre y cuando estos usuarios se hayan autenticado ante el sistema accediendo al mismo, momento en el que se almacena su certificado. El método que permite tal servicio es el siguiente.

public CertificadoInf[] solicitarCertificadoAbonados(String[] anagramasAbons)

En la invocación del anterior método será necesario que la Entidad Emisora indique los siguientes parámetros:

- AnagramasAbons: Es el conjunto de anagramas fiscales que son los identificadores unívocos de los usuarios, de los cuales queremos obtener su certificado.

Como resultado de la invocación del método se obtendrá un *array* de tipo *CertificadoInf* por cada anagrama consultado. En cada objeto devuelto vendrá el certificado y el anagrama al que va asociado. En el caso de devolver un objeto del *array* que fuera *null* significaría que el abonado no está dado de alta en el sistema.

Un ejemplo de código JAVA que permite la invocación de esta funcionalidad sería el siguiente:

```
Instanciamos el API de Entidades Emisoras
  notificaciones.cliente.api.MCSN mcsn = new notificaciones.cliente.api.MCSN("C:/mcsn.properties",true);
Enviamos la solicitud de obtención del certificado asociado al usuario Roberto Paz Robledo
 String[] ids = new String[1];
 Ids[0]= “09678456UPAZ ROBR”;
 CertificadoInf[] res = mcsn.solicitarCertificadoAbonados(ids);
Procesamos el resultado
 for (int i = 0; i < res.length; i++) {
   if (res[i] == null)
     logger.info("El usuario "+ids[i]+" NO está dado de alta en el sistema o no se encuentra su certificado");
  else
    logger.info("Se ha obtenido el certificado del usuario " + ids[i]);
 }
```

```
logger.info("Solicitud de Obtención realizada correctamente");
```

3.7.5 Información de remesas enviadas

A través del API suministrado, la Entidad Emisora podrá obtener información acerca de las remesas enviadas y las notificaciones contenidas en las mismas. Esta información podrá ser obtenida a través de un conjunto de funciones filtro que a continuación se indican:

> **Nota**
>
> Sólo con el método *obtenerInfRemesas(int[], boolean conAcuses)* existe la posibilidad de obtener los acuses generados por el sistema.

- Información acerca de las remesas enviadas a partir de sus códigos devueltos en el envío de remesa.

  Método:

  RemesaInf[] obtenerInfRemesas(int[] remesas, bolean conAcuses)

  El parámetro *conAcuses* indica si se desea obtener los acuses generados, por el sistema y por el abonado en la lectura de la notificación, en el *array* de *RemesaInf*

- Información acerca de las remesas asociadas a un servicio y procesadas entre una fecha inicial y final, que contengan notificaciones leídas por el Usuario.

  Método:

  RemesaInf[] obtenerInfRemesaConNotifLeidas(Date fechaIni, Date fechaFin, int cod_Suscripcion)

- Información acerca de las remesas asociadas a un servicio y procesadas entre una fecha inicial y final, que contengan notificaciones **no** leídas por el Usuario.

  Método:

  RemesaInf[] obtenerInfRemesaConNotifNoLeidas(Date fechaIni, Date fechaFin, int cod_Suscripcion, boolean cumplidoPlazo)

  El parámetro *cumplidoPlazo* es un modificador del método que permite, si es *true,* obtener información acerca de las remesas y notificaciones que no han sido leídas en el plazo máximo exigible de 10 días en el Real Decreto 209/2003 desde la recepción de la notificación, siendo notificada o avisada por correo electrónico o por el envío de un SMS dicha recepción. En el caso de que su valor sea *false,* se obtendrán todas las notificaciones no leídas, tanto rechazadas como no.

- Obtención de información acerca de las remesas entregadas al Sistema de Notificaciones y procesadas por el mismo entre una fecha inicial y final, y asociadas a una suscripción concreta de la entidad.

  Método:

  RemesaInf[] obtenerInfRemesa(Date fechaIni, Date fechaFin, int cod_Suscripcion)

- Información acerca de las remesas entregadas al Sistema de Notificaciones y procesadas por el mismo entre una fecha inicial y final, asociadas a una

suscripción concreta de la entidad, y que contienen notificaciones destinadas a un grupo de abonados.

Método:
RemesaInf[] obtenerInfRemesa(Date fechaIni, Date fechaFin, int cod_Servicio, Abonado[] abonados)

4. Integración de Notific@ en una aplicación de un Organismo

Como es lógico pensar el uso de este API requiere de un pequeño desarrollo de integración. Para llevar a cabo una integración exitosa será necesario tener presente los siguientes puntos:

1. **Suscripción de usuarios a servicios asociados.** Pueden existir dos opciones.
    a. Usuario suscrito mediante API. En este caso será necesario disponer de datos importantes acerca del usuario, como son:
        i. Anagrama fiscal largo, que permite identificar al usuario de forma unívoca. Esta cadena podrá ser obtenida invocando la utilidad *notificaciones.common.util.GeneradorIDUsuario* al que se le facilitará el nombre, el primer apellido, el segundo apellido y el NIF. **Es muy importante que todos estos datos coincidan en caracteres y espacios con los aparecidos en el certificado del usuario.**

        Existe otra posibilidad de obtener el anagrama fiscal largo, siempre y cuando se utilice el servidor de autenticación de la plataforma de @firma para autenticar a los usuarios que se conectan a la aplicación del Organismo. Entre los datos devueltos por esta plataforma se encuentra el anagrama fiscal obtenido a partir del certificado autenticado.
        ii. Nombre
        iii. Apellidos
        iv. Y opcionalmente Correo Electrónico y SMS, si se desea que el sistema automáticamente envíe un aviso de llegada de notificación.

Nota
Para la suscripción de un usuario mediante el API, es necesario que el usuario firme un formulario dando su consentimiento expreso de que desea que le notifiquen vía telemática todas las notificaciones generadas por un procedimiento.

    b. Usuario suscrito por iniciativa propia. En este caso solo existe un mecanismo para averiguar que usuarios están suscritos a un servicio y es invocando el método *AbonadoInf[] solicitarInformacionAltasAbonado(Date fechaIni, Date fechaFin,int cod_Servicio)*. Este método devolverá información

(Nombre, apellidos, anagrama fiscal) de todos los usuarios suscritos al servicio indicado.

Será responsabilidad de la aplicación almacenar en base de datos, con el objetivo de enviar en un futuro una remesa de notificaciones, el anagrama fiscal junto con otros datos de interés, de todos los usuarios suscritos a un servicio asociado, y comprobar periódicamente, que usuarios se han dado de baja del servicio asociado, invocando al método *AbonadoInf[] solicitarInformacionBajasAbonado(Date fechaIni, Date fechaFin,int cod_Servicio).*

2. **Envío de Remesa de Notificaciones.** Una vez que sabemos que usuarios están suscritos a un servicio asociado, bien porque los hemos suscritos a través del API, o bien porque se han suscrito ellos por iniciativa propia, podremos enviarles notificaciones, confeccionando una remesa.

> **Nota**
>
> Es importante antes de enviar una remesa con notificaciones, comprobar que los destinatarios están suscritos al servicio destino de la remesa. Para ello se podrá invocar al método *int[] solicitarEstadoAbonadosServicio(String[] anagramasAbons, int cod_Servicio).* Solo en aquellos destinatarios en los que devuelva '1', podrá enviarse una notificación. Para aquellos que devuelva '0' o '-1' será necesario darles de alta antes de enviarles una notificación.

Es responsabilidad de la aplicación almacenar en base de datos el código devuelto por el sistema en el envío de la remesa, ya que este código permitirá verificar posteriormente que la remesa ha sido procesada correctamente y que las notificaciones contenidas en la misma, han sido leídas o rechazadas por los usuarios o rechazadas por el sistema transcurridos el plazo de diez días desde la puesta a disposición y la no lectura por el destinatario.

Además es conveniente registrar las notificaciones que se envían, con el objetivo de, en un futuro, registrar cualquier evento que se produzca sobre la misma, ya sea por la puesta a disposición, como por su lectura o rechazo. Por ello es conveniente identificar la notificación en el origen, utilizando el método *setId(int)* de la clase *Notificacion*, ya que este identificador servirá para cuando se obtenga información de las remesas enviadas, hacer el casamiento y así poder identificar la notificación que se envío.

3. **Obtener información de remesas enviadas.** Tiene por objetivo verificar que el procesamiento de las remesas ha sido satisfactoria. Hay que tener presente que el procesamiento de una remesa se hace de forma asíncrona, lo que supondría una pérdida de tiempo verificar una remesa nada más enviarla.

   Tiempo para verificar el procesamiento de una remesa

   Lo idóneo, sería verificar una remesa enviada a partir de las **cinco horas** desde que se envío, permitiendo de esta manera dejar suficiente tiempo al sistema para procesarla adecuadamente. La forma de verificar su procesamiento sería invocando al método *RemesaInf[] obtenerInfRemesas(int[] remesas, bolean conAcuses)*

Por cada *remesaInf* existe un array de *NotificacionInf.* Cada *NotificacionInf* tiene un estado que es devuelto por el método *getEstado()* y que puede devolver cualquier constante definida en la clase *notificaciones.common.util. EstadoNotificacion.* Entre los estados que puede devolver se encuentran:

Estado	Descripción
0	Notificación puesta a disposición del abonado.
5	Notificación puesta a disposición y leída por el usuario.
6	Notificación puesta a disposición y rechazada por el usuario.
7	Notificación puesta a disposición y rechazada por el sistema automáticamente, por no lectura de la notificación por el usuario durante los diez días desde su puesta a disposición.
-1	Se ha enviado una notificación a un destinatario no dado de alta en el sistema.
-2	Se ha enviado una notificación a un destinatario no suscrito al servicio destino de la remesa.

Tiempo para verificar la lectura o rechazo de una notificación por el destinatario

Para verificar la lectura o rechazo de la notificación contenida en la remesa es necesario también invocar al anterior método, una vez transcurridos los **once días** desde que se envío la remesa. Esto permitirá que el sistema haya rechazado[2] automáticamente aquellas notificaciones que no hayan sido leídas o rechazadas por el usuario y por lo tanto se tendrá información fidedigna acerca del estado finalizado de todas las notificaciones enviadas, pudiendo estar estas en los siguientes estados posibles:

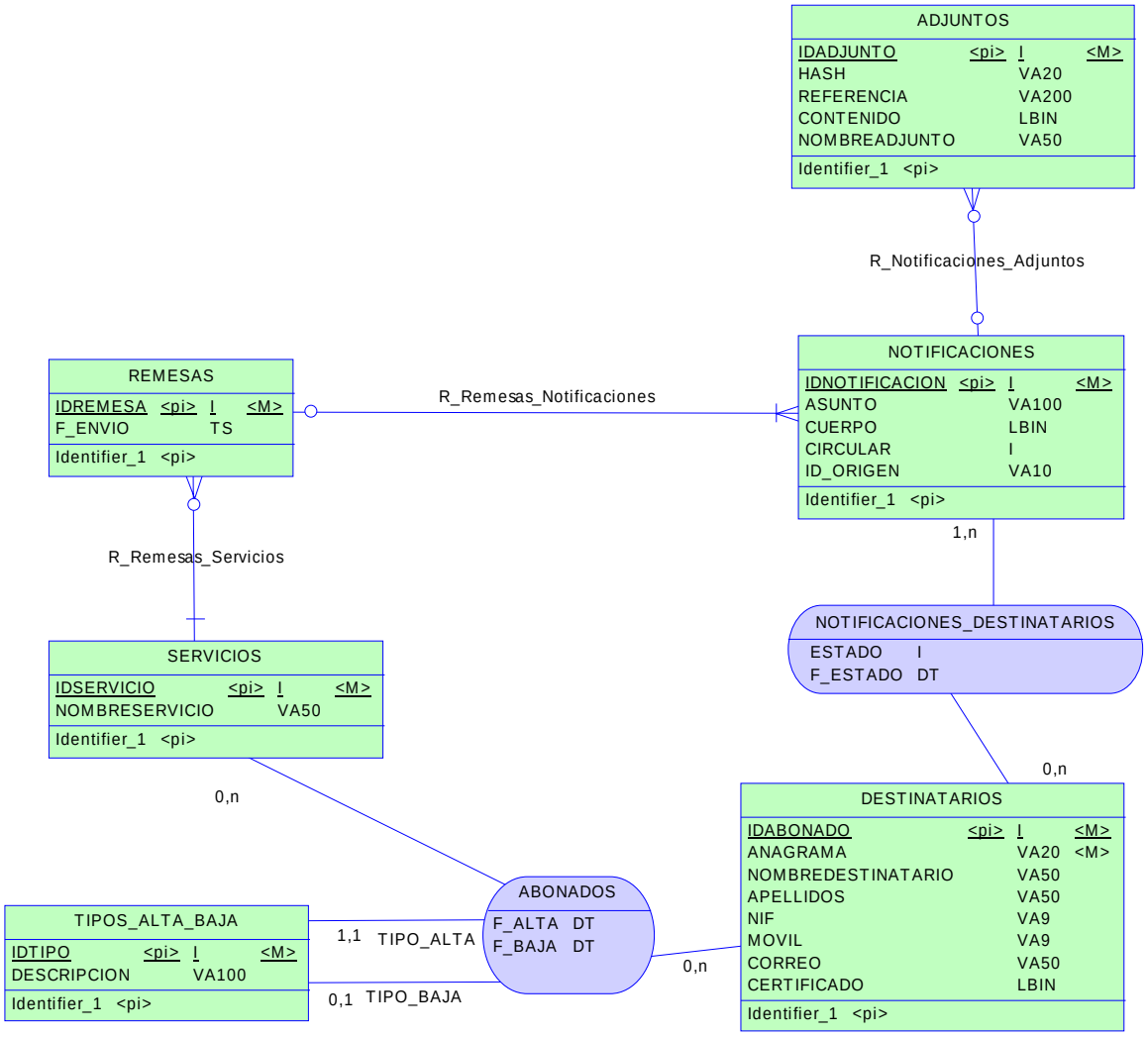
- Notificación leída por el usuario.
- Notificación rechazada por el usuario.
- Notificación rechazada por el sistema.

[2] Según el Real Decreto 209/2003 de 21 de Febrero, el usuario tendrá un plazo de diez días desde la puesta a disposición de la notificación para leerla o rechazarla, transcurridos estos diez días el sistema automáticamente rechazará la notificación haciendo que esta ya no pueda leerla el usuario.

4.1 Modelo Entidad-Relación

Con el objetivo de facilitar la integración de una aplicación en Notific@ se describe a continuación el modelo de entidad-relación que se debería utilizar. La descripción de cada una de las entidades que participan en el modelo vienen descritos en el siguiente punto.

El script de oracle para la creación de las tablas se encuentra en un fichero aparte denominado ***scriptDBClient.sql***



Nota

En el caso de que la Entidad emisora solo tenga un servicio asociado, la tabla SERVICIOS puede eliminarse trasladando los atributos de la tabla ABONADOS a la tabla DESTINATARIOS.

4.1.1 Entidades

ADJUNTOS

Entidad que representa a los archivos que se adjuntan a una notificación.

NOMBRE	TIPO	DESCRIPCION
IDADJUNTO	INTEGER	Identificador único del Adjunto (Clave Primaria)
HASH	VARCHAR(20)	Resumen SHA-1 en HEX del adjunto que lo identifica unívocamente
REFERENCIA	VARCHAR(200)	En el caso de que no se desee almacenar el adjunto en la base de datos, en este campo se indicará una referencia a su ubicación (ruta completa, ftp, http, etc).
CONTENIDO	BLOB	Contenido del fichero.
NOMBREADJUNTO	VARCHAR(50)	Nombre del adjunto que aparecerá en la notificación.

NOTIFICACIONES

Son las notificaciones que se agrupan y se envían a través de una remesa a Notific@.

NOMBRE	TIPO	DESCRIPCION
IDNOTIFICACION	INTEGER	Identificación univoca de la notificación en local.
ASUNTO	VARCHAR(100)	Asunto de la notificación
CUERPO	BLOB	Cuerpo de la notificación
CIRCULAR	INTEGER	‘0’ si la notificación es una notificación circular, es decir si es una notificación que se envía a todos los abonados suscritos a un servicio asociado, o ‘1’ si lo es.
ID_ORIGEN	VARCHAR(10)	Identificación de la notificación en origen. Va a servir para realizar el casamiento de la notificación enviada y la información devuelta por el sistema de notifica sobre el estado de la notificación

NOTIFICACIONES_DESTINATARIOS

Tabla de asociación que permite asociar una notificación a un destinatario en particular

NOMBRE	TIPO	DESCRIPCION
ESTADO	INTEGER	Estado de la notificación. '0': Puesta a disposición. '5': Puesta a disposición y leída. '6': Notificación rechazada por el usuario. '7': Notificación rechazada por el sistema por no lectura de la notificación por el destinatario durante los diez días posteriores de la puesta a disposición. '-1': Usuario no suscrito al servicio donde se envío la notificación. '-2': Usuario desconocido por el sistema.
F_ESTADO	DATE	Fecha de obtención del estado.

DESTINATARIOS

Representa a los destinatarios de las notificaciones

NOMBRE	TIPO	DESCRIPCION
IDABONADO	INTEGER	Identificador del destinatario en ciente
ANAGRAMA	VARCHAR(20)	Identificador del destinatario en Notific@
NOMBREDESTINATARIO	VARCHAR(50)	Nombre del destinatario
APELLIDOS	VARCHAR(50)	Apellidos
NIF	VARCHAR(9)	Número de Identificación Fiscal
MOVIL	VARCHAR(9)	Móvil del destinatario donde recibirá avisos de llegada de notificación
CORREO	VARCHAR(50)	Correo electrónico del destinatario donde recibirá avisos de llegada de notificación
CERTIFICADO	BLOB	Certificado (Clave pública) utilizado para cifrar notificaciones.

SERVICIOS

Esta tabla no sería necesaria en el caso de que la Entidad Emisora solo tuviera un servicio asociado al que notificar.

NOMBRE	TIPO	DESCRIPCION
IDSERVICIO	INTEGER	Identificador del servicio en Notific@
NOMBRESERVICIO	VARCHAR(50)	Nombre del servicio en Notific@

TIPOS_ALTA_BAJA

Representa a los distintos tipos de solicitud de alta o baja que se puede tener:

- Vía formulario Web.
- Por teléfono.
- Presentando un impreso de solicitud en ventanilla.
- Etc.

NOMBRE	TIPO	DESCRIPCION
IDTIPO	INTEGER	Identificador secuencial del tipo solicitud de alta o baja
DESCRIPCION	VARCHAR(100)	Descripción del método seguido para realizar el alta o baja.

REMESAS

Representa a las remesas (conjunto de notificaciones) que se envían a Notific@.

NOMBRE	TIPO	DESCRIPCION
IDREMESA	INTEGER	Identificador de la remesa devuelta por el sistema en el envío.
F_ENVIO	DATE	Fecha de envío de la remesa de notificaciones

ABONADOS (Tabla relación)

Representa un usuario dado de alta en Notific@ y en un servicio asociado a la Entidad Emisora.

NOMBRE	TIPO	DESCRIPCION

F_ALTA	DATE	Fecha de alta de usuario en el servicio
F_BAJA	DATE	Fecha de baja del usuario en el servicio
TIPO_ALTA	INTEGER	Método empleado por el usuario para solicitar el alta en servicio (Formulario Web, impreso de solicitud, teléfono, etc).
TIPO_BAJA	INTEGER	Método empleado por el usuario para solicitar la baja del servicio (Formulario Web, impreso de solicitud, teléfono, etc).