5. Push Registry

En la especificación MIDP 2.0 se añaden nuevas características y facilidades con respecto a la versión 1.0 del perfil. Entre ellas destacan una interfaz de usuario mejorada, funcionalidad para multimedia y juegos, instalación y actualización de MIDlets por el aire (OTA), o un nuevo modelo de seguridad extremo a extremo. Otra de estas nuevas características será *Push Registry*, que veremos en profundidad en este proyecto.

La característica *Push Registry* permite a un MIDlet configurarse a sí mismo de forma que permita ser iniciado autómaticamente, sin que el usuario tenga que hacerlo manualmente como es común. Este inicio de forma automática puede tener lugar de dos maneras: mediante una conexión entrante o cuando lo indique un temporizador previamente establecido. Esto permitirá por ejemplo que cuando se reciba un nuevo correo electrónico se arranque automáticamente una aplicación para procesarlo, o que periódicamente se inicie una aplicación que se sincronice con un servidor. Por lo tanto el ciclo de vida que habíamos visto hasta ahora (figura 3.3) quedará modificado, de forma que el inicio del MIDlet pueda ser ejecutado de varias maneras como muestra la figura 5.1.

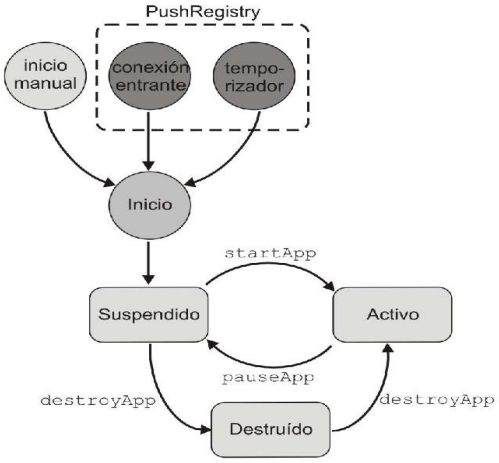

Figura 5.1 Ciclo de vida con PushRegistry

Push Registry forma parte del AMS, que como ya se citó es el software responsable del ciclo de vida del MIDlet*, y* proporciona por un lado la API necesaria para su manejo dentro del AMS, y posee por otro lado una lista de todos los registros. En la figura 5.2 se ilustra con más detalle los distintos elementos de *Push Registry*.

En el caso de inicio de MIDlet por conexiones entrantes, éstas pueden ser de muchos tipos diferentes. De este modo podrán ser de tipo mensaje (SMS), flujo (como *sockets* TCP) o basadas en paquetes (como datagramas). En particular este proyecto se centrará en *Push Registry* de conexiones entrantes sobre Bluetooth, que es lo que se verá de aquí en adelante.

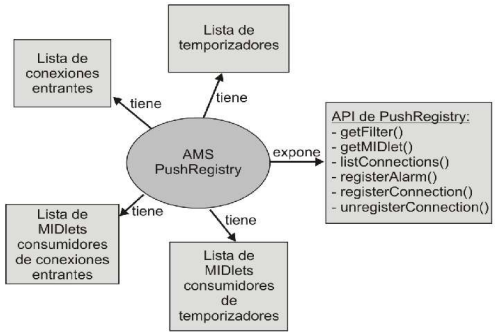

Figura 5.2 Elementos de PushRegistry

5.1. Ejemplos de casos de uso

Con el fin de ver la utilidad y ventajas de esta característica que incorpora el MIDP 2.0, a continuación se detallan algunos casos típicos de uso desde el punto de vista de consumidor final.

5.1.1. Inicio más sencillo de aplicaciones

Supongamos un juego Java instalado en un teléfono con atributos Push en su fichero JAD/JAR. El juego será entonces visible y conectable por otros dispositivos aunque en el teléfono no esté la aplicación abierta y se estén usando otras aplicaciones. En el caso que otro dispositivo trate de establecer una conexión para iniciar el juego de forma multijugador, el teléfono mostrará una pantalla que preguntará al usuario si desea que se inicie el juego con la conexión entrante. Si el usuario acepta el juego se iniciará y podrá jugar de forma multijugador con el otro dispositivo. En cualquier caso, el usuario también podrá rechazar la petición y seguir usando el teléfono normalmente. Esto supone una manera más sencilla de iniciar una aplicación Java de este tipo.

5.1.2. Inicio instantáneo de aplicaciones

Otra aplicación puede ser la de recibir mensajes instantáneos via Bluetooth. Podemos pensar en una aplicación de mensajes configurada de forma que sea visible y conectable por otros dispositivos. Aun más, el *service record* correspondiente a este servicio almacenado en la SDDB, puede contener una pequeña página personal, accesible en todo momento sin tener que tener iniciada la aplicación en todo momento.

Cualquier otro dispositivo que use el cliente de mensajes podrá detectar al primer dispositivo e incluso tener acceso a la página personal. Este segundo dispositivo podrá enviar un mensaje al primero. En el primer teléfono aparecerá entonces un aviso que pide confirmación para iniciar el cliente de mensajes debido a una conexión entrante. En el caso que se acepte se recibirá el mensaje. También se podrá cancelar el inicio de la aplicación con lo que el mensaje será

automáticamente descartado. Por otro lado, también se podrán tener listas de exclusión, en las que se podrán añadir dispositivos a los que se ignorará directamente, y la aplicación no reaccionará en absoluto por un intento de conexión entrante de estos dispositivos.

Esto permite que en un extremo se puedan utilizar aplicaciones de comunicaciones de forma instantánea, sin que tenga que estar el otro extremo esperando con la aplicación iniciada.

El inicio de aplicaciones de forma remota puede ser también muy interesante a la hora de efectuar operaciones de sincronización o mantenimiento. Es decir, gracias a *PushRegistry* se puede tener una aplicación que se inicie automáticamente para realizar determinadas tareas, sin necesitar la intervención por parte del usuario. Supongamos como ejemplo una aplicación de este tipo iniciada periódicamente por un servidor remoto para actualizarla.

5.1.3. Aplicaciones Punto a Multipunto

En las aplicaciones Punto a Multipunto se establece una comunicación entre más de dos dispositivos al mismo tiempo. Por ejemplo cuando se establece una comunicación con otro dispositivo para iniciar un juego como el del primer escenario, mientras se tiene una comunicación con un auricular Bluetooth inalámbrico. El problema que puede suponer es que en una de las comunicaciones se esté usando la interfaz de usuario, impidiendo así el inicio manual de cualquier otra aplicación.

Supongamos el siguiente ejemplo de comunicación Punto a Multipunto. Se desea enviar un archivo de un dispositivo a otro, para lo que se establece un enlace entre ellos. Durante la transmisión, uno de los dos desea ejecutar una aplicación que se comunica con un tercer dispositivo. El problema es que el usuario no podrá iniciar esta aplicación porque la interfaz de usuario está ocupada mostrando el estado de la transferencia del archivo. Sin embargo, mediante *PushRegistry*, el tercer dispositivo al conectarse al segundo haría que se iniciase la aplicación sin tener que usar la interfaz de usuario para iniciarla manualmente.

5.2. La API de PushRegistry

Una vez vistos algunos posibles casos de uso y ventajas que *PushRegistry* aporta, veremos que es lo que nos proporciona la API de Java para utilizar esta característica.

Los servicios vistos en el capítulo 4, en el que se explica la API de Java para Bluetooth se denominan *run-before-connect*, pues presuponen la ejecución de la aplicación antes de comenzar cualquier proceso de conexión. Por esto mismo el servicio no estará visible hasta que se inicie la aplicación y se llame al método `acceptAndOpen()`, es decir, hasta que se inicie el servidor (ver figura 4.3).

Los servicios que hacen uso de la característica *PushRegistry* se denominan sin embargo servicios *connect-anytime*, pues permiten la conexión al servicio en cualquier momento. De este modo, el ciclo de vida de estos servicios será bien distinto pues el servicio tiene que estar disponible antes de iniciar la aplicación. Como vemos en la figura 5.3 comenzará a estar disponible desde el momento en el que se registre, y a partir de entonces será posible su conexión.

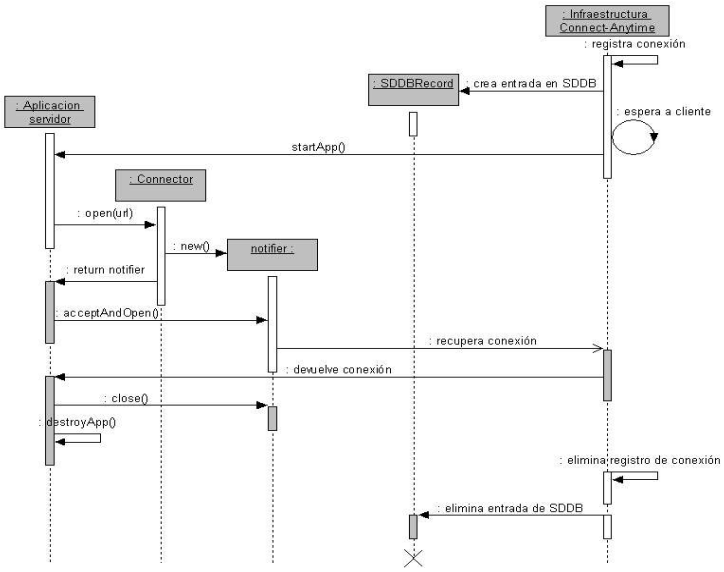

Figura 5.3 Ciclo de vida de servicio connect-anytime

La especificación de JABWT no solo da soporte a los típicos servicios *run-before-connect*, sino que también permite utilizar servicios que utilicen la característica *PushRegistry*. Para ello el MIDP 2.0 proporciona la clase `PushRegistry` que forma parte del paquete `javax.microedition.io`. Esta clase ofrece todos los métodos relacionados con los procesos de *push*. A continuación, se exponen las distintas operaciones que se pueden llevar a cabo y mediante qué métodos. Se explicará el funcionamiento de éstos, incluyendo las excepciones que pueden lanzar.

5.2.1. Registrar conexión

Para que un MIDlet pueda ser iniciado mediante una conexión entrante, debera registrarse en el *Push Registry*. Esto se podrá realizar de dos formas distintas: dinámica o estáticamente. Veamos a continuación en que consiste cada una.

a) Registro estático

El registro se podrá realizar de manera estática al instalar el MIDlet. Para ello habrá que añadir un atributo `MIDlet-Push` al arhivo JAD y manifiesto del JAR. Con este atributo la conexión será registrada al instalar el MIDlet. La estructura que sigue es la siguiente:

```
MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>
```

donde:

- `MIDlet-Push-<n>` es el nombre de la propiedad que indica que esa entrada del fichero corresponde a un registro *push*, donde `<n>` será un entero que comienza en 1 y va tomando valores sucesivos. Este entero se emplea ya que es posible registrar más de una conexión, donde cada una tendrá una entrada con un valor diferente.
- `<ConnectionURL>` será la cadena que se utiliza para la conexión entrante a registrar. En el caso de una conexión Bluetooth un ejemplo sería

  ```
  btspp://localhost:1231242432434AAAABB
  ```

- `<MIDletClassName>` debe ser el nombre de la clase del MIDlet a iniciar cuando se detecte la conexión entrante. El nombre del MIDlet debe ser el mismo que aparezca en el fichero JAD y manifiesto del JAR en una entrada `MIDlet-<n>`.
- `<AllowedSender>` es un filtro que se utiliza para indicar la dirección de los dispositivos autorizados a iniciar el MIDlet mediante la conexión remota. El carácter `"*"` sustituye a cualquier cadena, por lo que permite la conexión de cualquier dispositivo. Por otro lado `"?"` podrá sustituir a cualquier carácter.

En el siguiente ejemplo de archivo JAD, se ve como en la última línea se registra el MIDlet `HolaMundo`, para que se inicie cuando se establezca la conexión Bluetooth especificada. Debido a que en el campo destinado al filtro se ha especificado `"*"` se permite la conexión desde cualquier dispositivo.

```
MIDlet-Name: Hola Mundo
MIDlet-Version: 0.0.1
MIDlet-Vendor: MyCompany
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-Jar-URL: HolaMundo.jar
MIDlet-Jar-Size: 1231
MIDlet-1: HolaMundo, , HolaMundo
MIDlet-Push-1: btspp://localhost:123456789AB, HolaMundo, *
```

b) Registro dinámico

Ya se ha visto como registrar una conexión durante la instalación del MIDlet. Además de este registro estático existe la posibilidad de realizar el registro dinámicamente durante la ejecución del MIDlet. Esto es posible mediante el método `registerConnection()` perteneciente a la clase `PushRegistry`. Los parámetros que recibe son la cadena de conexión, el nombre del MIDlet responsable de la conexión, y el filtro que permite restringir las conexiones entrantes de determinados clientes. Como vemos los parámetros son los mismos que para realizar el registro estático mediante el atributo `MIDlet-Push-<n>`. Como es de esperar los parámetros adoptarán además la misma forma que en el registro estático.

Este método no devolverá ningún valor, sin embargo si podrá lanzar varios tipos de excepciones:

- `IllegalArgumentException` si la cadena de conexión y/o del filtro no son válidas.
- `ConnectionNotFoundException` si el sistema no soporta el tipo de conexión entrante indicado.
- `ClassNotFoundException` si la clase especificada como MIDlet a registrar no se encuentra. O lo que es lo mismo, si no se encuentra la entrada `MIDlet-<n>` con el nombre de clase correspondiente en el fichero JAD.
- `IOException` si la conexión ya está registrada, o si no hay recursos suficientes para llevar a cabo la operación.
- `SecurityException` si el MIDlet especificado no está autorizado para registrar una conexión.

El siguiente código realizaría lo mismo que vimos en el ejemplo de registro estático pero de forma dinámica.

```
...
// Nombre del MIDlet
String midletClassName = "HolaMundo";
// Cadena de conexion
String connectionUrl = "btspp://localhost:234512324AAABBB";
// Filtro. Con "*" permite la conexión desde cualquier dispositivo
String pushFilter = "*";

try {
    PushRegistry.registerConnection(connectionUrl, midletClassName, pushFilter);
} catch (IOException e) {
    form.append("\nIOException, conexion posiblemente ya registrada");
} catch (ClassNotFoundException e) {
    form.append("\nNombre de MIDlet no encontrado");
} catch (SecurityException e) {
    form.append("\nPermisos insuficientes");
}
...
```

5.2.2. Eliminar registros

Mediante el método `unregisterConnection()` se elimina dinámicamente el registro de una conexión. Como parámetro tomará la conexión a eliminar, que será la misma cadena que se empleó en `registerConnection()`. El método devolverá un `boolean`, que será `true` si el registro se pudo eliminar sin problemas, o `false` en caso contrario o si el argumento es `null`. Además podrá lanzar `SecurityException` si la conexión fue registrada por otro MIDlet. Eliminar un registro es tan sencillo como se muestra a continuación.

```
...
try {
    boolean status;
    status = PushRegistry.unregisterConnection(url);
}
catch(SecurityException e) {
    form.append("\nPermisos insuficientes");
}
...
```

Además de esto existe, análogamente que a la hora de realizar el registro, lo que se denominaría eliminación estática del registro. Esto se hace automáticamente al desinstalar el MIDlet del dispositivo, lo que quiere decir que al realizar la desinstalación se eliminarán todos los registros realizados por el MIDlet.

5.2.3. Listar conexiones

El método `listConnections()` devuelve una lista de las conexiones registradas por el MIDlet que lo llama. Si se le pasa como parámetro `false`, devolverá absolutamente todas las conexiones registradas por el MIDlet. Si por el contrario el argumento es `true`, sólo devolverá las conexiones registradas que están establecidas en ese momento. Por lo tanto este método será bastante útil, porque llamándolo al principio de la ejecución del MIDlet con `true` como argumento, nos permitirá saber si el MIDlet ha sido iniciado debido a una conexión entrante. Esto mismo es lo que se hace en el siguiente ejemplo. En este suponemos que la conexión no se ha establecido en el MIDlet, por lo que si hay una conexión activa es la que ha provocado que se inicie el propio MIDlet.

```
...
// Lista las conexiones activas en este momento
String[] connections = PushRegistry.listConnections(true);
if (connections != null && connections.length > 0) {
    for (int i=0; i < connections.length; i++) {
        form.append("\nMIDlet iniciado por conexion entrante");
    }
}
...
```

5.2.4. Información sobre la conexión

La clase `PushRegistry` proporciona dos métodos para obtener información sobre la conexión *push*. Esta información se estableció a la hora de registrar la conexión en cuestión. Por un lado el método `getMIDlet()` devuelve el MIDlet registrado para una conexión determinada, que se le pasará como parámetro. Por otro `getFilter()` devuelve el filtro asignado a una conexión determinada. Ambos métodos tendrán como parámetro la cadena de conexión. Si la conexión no está registrada o el argumento es `null`, devolverán `null`.

El siguiente código de ejemplo muestra por pantalla información referente a todos los MIDlets registrados en el *PushRegistry*.

```
...
// Obtiene lista de todas las conexiones registradas
String[] connections = PushRegistry.listConnections(false);

if (connections != null && connections.length > 0) {
    for (int i=0; i < connections.length; i++) {
        // obtiene nombre del MIDlet registrado
        String midlet = PushRegistry.getMIDlet(connections[i]);
        // obtiene filtro
        String filter = PushRegistry.getFilter(connections[i]);
        // Muestra la información
        form.append(connections[i] + " " + midlet + " " + filter);
    }
}
...
```

5.2.5. Registrar alarma

Además de por una conexión entrante, un MIDlet podrá ser automáticamente iniciado en un momento determinado registrando previamente lo que se denomina una alarma. Aunque no sea el objetivo principal de este proyecto, a continuación se describe brevemente como se realiza esto.

El registro de una alarma se podrá realizar mediante el método `registerAlarm()`. Tomará como parámetros el nombre del MIDlet a iniciar, que deberá seguir las consideraciones mencionadas para `registerConnection()`, y un temporizador que indica cuando debe ser iniciado el MIDlet. Si ya había una alarma registrada será sobreescrita y el valor devuelto será el momento para el que estaba registrada la alarma. En caso contrario el método devolverá `0`. Tanto el tiempo que devuelve el método como el que se le pasa como parámetro deberán seguir el formato de `Date.getTime()`.

Las excepciones `ConnectionNotFoundException`, `ClassNotFoundException`, o `SecurityException`, pueden tener lugar por causas análogas que al utilizar el método `registerConnection()`.

Este método permitirá que un MIDlet se inicie periódicamente. El siguiente ejemplo ilustra como hacerlo. Una vez iniciado el MIDlet, se registrará una alarma para que se inicie de nuevo al cierto tiempo. Esto se realizará justo antes de terminar el MIDlet en el método `destroyApp()`.

```
...
// metodo que ejecuta el MIDlet antes de ser destruido
public void destroyApp(boolean uc) throws MIDletStateChangeException {
    ...
    // Registra la alarma antes de salir
    try {
        String cn = "MyPushMIDlet";
        long deltatime = 1000*60*5;  // 5 minutos
        // Obtiene el tiempo actual
        Date alarm = new Date();
        long t = PushRegistry.registerAlarm(cn, alarm.getTime() + deltatime);
    } catch (ClassNotFoundException e) {
```

```
            form.append("\nNombre de MIDlet no encontrado");
        } catch (SecurityException e) {
            form.append("\nPermisos insuficientes");
        }
    }
    ...
```

5.2.6. Seguridad de PushRegistry

El hecho de que una aplicación pueda ser iniciada de forma remota, hace que se incremente el riesgo de la ejecución de una aplicación no deseada. Esto hace que la seguridad sea un aspecto muy importante a la hora de utilizar *PushRegistry*. Por ello el perfil MIDP 2.0 introduce dos nuevos conceptos de seguridad, que son la firma de aplicación y los dominios privilegiados.

La firma de aplicación le permite determinar a la plataforma si confiar en una aplicación, basándose en el origen e integridad de ésta y en el uso del certificado digital X.509. Los dominios de protección permiten a la plataforma definir una serie de normas para restringir el acceso de determinadas APIs.

En múltiples plataformas las operaciones de redes y *push* serán operaciones restringidas, por lo que las aplicaciones tendrán que solicitar previamente el uso de las APIs. Si se trata de usar una operación que está restringida sin haber pedido el permiso necesario el sistema lanzará una excepción del tipo `SecurityException`.

Los permisos se solicitan en el archivo JAD y manifiesto del JAR, creando una entrada del tipo `MIDlet-Pemissions`. Los nombres de los permisos seguirán la convención típica usada para los paquetes de Java. Por ejemplo, en el caso que se quiera usar `PushRegistry` y en la plataforma tenga un acceso restringido habrá que incluir la línea siguiente:

```
MIDlet-Permissions: javax.microedition.io.PushRegistry
```